Strategic Hard-Negative Selection for Efficient Fine-Tuning of Dense Retrievers

Aiman Abdul Wahab



Delft University of Technology

TU Delft

Strategic Hard-Negative Selection for Efficient Fine-Tuning of Dense Retrievers

by

Aiman Abdul Wahab

to obtain the degree of
Master of Science
in
Data Science & Artificial Intelligence Technology
at the Delft University of Technology
to be defended publicly on Wednesday November 25th, 2026 at 14:00 PM.

Student number:	5256852
Project Duration:	November 9, 2024 - November 25, 2026
Thesis committee:	Prof. Dr. ir. A. Anand, TU Delft supervisor
	Dr. H.S. Hung, TU Delft

Web Information Systems Group
Faculty EEMCS, Delft University of Technology
Delft, The Netherlands
www.ewi.tudelft.nl

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

This thesis concludes my Master's journey in Data Science and Artificial Intelligence Technology at TU Delft. Through it, I learned to approach research more effectively, from reading the literature to designing experiments. This work was also very technical, as I learned to build fine-tuning pipelines, including preprocessing the data and evaluating the trained models.

I would like to express my gratitude to Dr. Avishek Anand for supervising me throughout this process, for their guidance, feedback, and the weekly research group meetings, where I got to learn more about the latest work in this field. I would also like to thank my peers in the research group for the discussions and time spent together, and my family and friends for their constant support.

Aiman Abdul Wahab
Delft, November 2026

Abstract

Dense retrievers are often trained using contrastive learning to differentiate a positive document from a negative one. The harder the negative, the better the training signal and thus the better the distinction. In-batch training takes its negatives from other queries in the same batch, while cross-batch increases the negative document size by using several batches of them. These documents, however, are most of the time easy to tell apart, and larger batches come at the cost of more GPU memory. ANCE finds harder negatives by periodically re-encoding the full corpus with a new checkpoint of the model, which is expensive to do on limited hardware. This thesis investigates whether hard negatives can be found without re-encoding the whole corpus. GRASS is then proposed, which mines documents from a small cache that periodically gets updated and refreshed, and uses the model's uncertainty in the selection of hard negatives. After fine-tuning the pretrained retriever BGE-M3, GRASS scored on par with ANCE on the 12 BRIGHT domains and higher than cross-batch and in-batch, while each mining round took about a third of the time of an ANCE round.

Contents

Preface i

Summary ii

1 Introduction 1

2 Literature Review 4

3 Methodology 10
3.1 Data, Preprocessing, and Evaluation . . . 10
3.1.1 Base Models . . . 10
3.1.2 Datasets . . . 11
3.1.3 Training-Data Composition and Preprocessing . . . 12
3.1.4 Data Formatting and Artifacts . . . 13
3.1.5 Document Segmentation (MaxP) . . . 14
3.1.6 Token Limits and Coverage . . . 14
3.1.7 Evaluation Pipeline . . . 14
3.2 Training and Negative-Selection Strategies . . . 16
3.2.1 Contrastive Loss for the BRIGHT Experiments . . . 16
3.2.2 Batch Construction . . . 16
3.2.3 In-Batch Negatives . . . 17
3.2.4 Cross-Batch Negatives . . . 17
3.2.5 Asynchronous ANCE training. . . . 17
3.2.6 GRASS . . . 18

4 Experimental Setup 23
4.1 Experiments . . . 23
4.2 Computational Environment . . . 23
4.2.1 Hardware: DelftBlue Cluster . . . 23
4.2.2 Software and Libraries . . . 24
4.3 Training Settings . . . 24
4.3.1 Shared Training Setup . . . 24
4.3.2 In-Batch . . . 24
4.3.3 Cross-Batch . . . 24
4.3.4 ANCE for BRIGHT . . . 25
4.3.5 GRASS . . . 25
4.3.6 MS MARCO ANCE Validation . . . 25
4.4 Comparisons . . . 25

5 Results 27
5.1 Overall Retrieval Performance . . . 27
5.2 Performance Across BRIGHT Domains . . . 27
5.3 Paired Comparisons Between Training Strategies . . . 28
5.4 Effect of Uncertainty-Guided Selection . . . 28
5.4.1 Retrieval Performance Across Uncertainty Weights . . . 28
5.4.2 Effect on Negative Selection . . . 28
5.5 Training and Mining Efficiency . . . 29
5.6 MS MARCO ANCE Reference Comparison . . . 30

6 Discussion 31
6.1 Main Findings . . . 31
6.2 Sample-Efficient Training Without Large Batches or Global Mining . . . 31

6.2.1 Cache-Based Mining Without Large Batches 31
6.2.2 Uncertainty-Guided Negative Selection 32
6.3 Training Cost and Retrieval Performance 32
6.3.1 Measured Cost at Competitive Retrieval 32
6.3.2 Expected Training Time on a Single GPU and Larger Corpora 33
6.4 Robustness Across Domains 33
6.5 Limitations 34
6.6 Disclosure 34

7 Conclusion **35**

References **36**

1

Introduction

Background & Motivation

Retrieval systems serve as the foundation in the information domain, used in diverse applications such as open-domain question answering (OpenQA), fact verification, and recommender systems. BM25 and TF-IDF have historically been used for this domain but suffer from vocabulary mismatch and are highly sensitive to spelling errors.

This motivated the exploration of Dense Retrieval (DR) by mapping queries and documents into a continuous semantic vector space. While various architectures have been proposed, bi-encoders are widely adopted for large-scale retrieval because queries and documents are encoded independently, enabling efficient similarity search over pre-computed document representations [22]. Most dense retrieval models use contrastive learning to distinguish between relevant and irrelevant documents based on semantic similarity.

Early dense retrieval models, such as Dense Passage Retrieval (DPR), demonstrated that dense encoders could outperform sparse methods, yet they exhibit significant limitations in domain transferability. The primary reason for this is a reliance on 'easy' in-batch negatives during training, which fail to provide enough signal for the model to learn nuanced semantic differences. Although researchers attempted to fix this by incorporating lexical hard negatives via BM25, this often forced dense models to mimic sparse retrieval, thus inheriting the same lexical biases they were meant to overcome [28].

Additionally, these models' performance is limited by a scarcity of high-quality labeled data in datasets such as MSMARCO, where the vast majority of the millions of passages remain unlabeled. This lack of exhaustive labeling means that many top-retrieved passages appear to be hard negatives but are actually relevant to the query (false positives), introducing noise that hinders the model's ability to reach its full capacity [20].

The deficiency motivated the development of current state-of-the-art (SOTA) models such as ANCE and RocketQA, which utilize more sophisticated, resource-intensive training pipelines. These approaches incorporate globally mined hard negatives or multi-stage distillation processes to force the encoder to learn more robust and discriminative representations.

However, these improvements come at a high computational cost. Models such as ANCE and RocketQA rely on resource-intensive pipelines involving very large batch sizes and frequent index refreshes. This demands large computational resources and long training times, making it difficult to fine-tune or reproduce under limited hardware. This gap motivates the core objective of this project: investigating sample-efficient training methods. By exploring bottleneck architectures and more efficient sampling strategies, this research aims to achieve high-recall dense retrieval without requiring massive hardware clusters or complex pipelines.

Problem Description

Current dense retrieval models are constrained by a trade-off between signal quality and computational overhead. This bottleneck is defined by three primary challenges:

1. **Weak Negatives:** Most dense models perform contrastive learning by utilizing in-batch negatives; randomly sampled documents that are typically unrelated to the query both lexically and semantically. While they are cheap to compute, they limit the model's ability to generalize because it is never challenged by nearly similar documents in the corpus. Once this simple separation is achieved, the model often converges to a suboptimal solution because the training signal is weak. To achieve high retrieval accuracy, the model must instead learn more discriminative embeddings. One way to encourage this is through "hard negatives": documents that are semantically similar to the query but actually irrelevant. These samples force the model to resolve subtle semantic differences, which is necessary to correctly rank relevant passages during global inference across millions of documents.
2. **Global Negative Mining:** To combat this, models are exposed to "hard" negatives: irrelevant passages that the model struggles to distinguish from relevant ones. These negatives can be assigned initially in the dataset alongside the positive document via lexical methods like BM25, or by using LLMs to generate synthetic challenging examples. However, as training progresses, these samples become stale. This necessitates dynamically updating negatives over time, using the current model's state to mine more difficult passages from the entire corpus. This, however, requires constant re-indexing of the entire corpus and complex pipelines, making state-of-the-art (SOTA) dense retrieval difficult for researchers with limited hardware or those needing models to adapt quickly to new domains.
3. **False Negatives & Lack of Dataset Quality:** Another challenge is the infeasibility of completely annotating every candidate passage in a large corpus. For instance, in the MSMARCO dataset, questions have an average of only 1.1 annotated positives despite a collection of 8.8 million passages. When researchers attempt to improve signal quality by sampling "hard negatives" from top-retrieved results, they frequently encounter unlabeled positives, causing many false positives during inference. To solve this, SOTA models like RocketQA and ReasonIR employ "denoising" or "synthetic labeling" using Cross-Encoders or LLMs to verify and relabel these samples. However, this creates a significant computational barrier, as running these billion-parameter models over millions of query-document pairs is only possible with expensive hardware.

The goal of this research is to investigate sample-efficient training strategies that provide the benefits of hard negative mining without the prohibitive costs of frequent global refreshes or heavy computational pipelines. Furthermore, this work focuses on limited fine-tuning settings, aiming to reduce the massive scale of datasets and hardware typically required to adapt models to new domains. By addressing these resource constraints, the project seeks to make fine-tuning more accessible and adaptable.

Research Questions

To address the identified bottlenecks and achieve the goal of resource-efficient dense retrieval, this project seeks to answer the following research questions:

1. **RQ1:** How can dense retrieval models be trained sample-efficiently to learn discriminative query–document representations without relying on large batch sizes or frequent global hard negative mining?
2. **RQ2:** To what extent can lightweight hard-negative sampling strategies reduce the computational cost and training time of dense retrievers while maintaining competitive retrieval performance in limited fine-tuning and low-resource settings?

Scientific Contributions

The thesis makes the following contributions:

- **A lightweight hard-negative mining method for dense retrieval.** GRASS was designed and implemented to mine hard negatives during training without re-encoding the corpus. Negatives

are selected from a bounded cache of document embeddings, maintained based on how useful its entries are as negatives. Mining runs asynchronously from training, and the training queries to revisit are chosen by how hard their mined negatives were.

- **Evaluation of training cost and retrieval performance on reasoning-intensive retrieval.** GRASS was compared with in-batch, cross-batch, and ANCE-style training with fine-tuning BGE-M3 and evaluating on BRIGHT. GRASS used half the training updates and less GPU time than ANCE-style mining, while reaching retrieval performance close to it.
- **Analysis of uncertainty-guided negative selection and query selection.** MC dropout uncertainty was evaluated as a criterion for selecting hard negatives, and the score gap between a query's negative and positive as a criterion for selecting training queries. Uncertainty changed which negatives were selected but did not consistently improve retrieval.

Thesis Outline

Chapter 2 reviews prior work on dense retrieval training, hard-negative mining, training-example prioritization, uncertainty estimation, and training under limited GPU memory. Chapter 3 describes the data, the BRIGHT evaluation, the baseline training strategies, and GRASS. Chapter 4 details the experimental setup, including the hardware, training configurations, and budgets. Chapter 5 reports retrieval performance, the effects of uncertainty and query selection, and training cost. Chapter 6 discusses these results and their limitations, and Chapter 7 concludes by directly answering the research questions.

2

Literature Review

Sparse to Dense Retrieval Passage

For years, retrieval systems have relied primarily on BM25 and other sparse algorithms. These methods remain popular because they are computationally efficient, easy to implement, and highly effective at matching exact keywords between a query and a document. Over time, researchers explored ways to overcome the limitations of keyword matching in sparse retrievers. The benefits of deep learning techniques were first explored in later-stage pipeline components, such as re-rankers and Machine Reading Comprehension (MRC). Nogueira and Cho (2019) used BERT_LARGE as a passage ranker and compared it with existing approaches such as BM25, IRNet, and KNRM. Their model achieved an MRR@10 of 36.5, representing a relative improvement of 27% over IRNet (27.9) and BM25 (16.7) [18].

Despite significant performance gains, these “expensive” cross-encoder models are not feasible for retrieval over large corpora, as they are non-decomposable and incur high computational latency. Earlier work, such as ORQA, also relied on the Inverse Cloze Task (ICT) for pretraining, which introduced additional computational overhead. Karpukhin et al. (2020) addressed these limitations with Dense Passage Retrieval (DPR) by adopting a bi-encoder architecture. In this setup, queries and documents are encoded independently and mapped into a shared vector space, enabling efficient similarity search over millions of documents using approximate nearest neighbor indexing methods such as FAISS. When evaluated in the Natural Questions (NQ) dataset using top-20 retrieval precision, DPR achieved a score of 78.5, significantly higher than the 59.1 achieved by BM25. Nonetheless, DPR still suffered from technical flaws such as the use of ’easy’ negatives, the false negative problem, and even performed worse than BM25 in datasets with high lexical overlap, such as SQuAD v1.1 [22].

Negative Mining and high-complex pipelines

To address the flaws of DPR, subsequent research shifted towards more sophisticated negative mining techniques and high-complexity training pipelines. The researchers aimed to narrow down the training-inference discrepancy and resolve the false negative problem by introducing mechanisms to select more informative training signals from the entire corpus.

ANCE (Approximate Nearest Neighbour Contrastive Estimation) mechanism involves two simultaneous asynchronous processes. The first process, the trainer, uses the index to retrieve the hard negatives, while the inferencer uses the latest trained checkpoint to update this indexing in order to ensure that the trainer does not use stale negatives as the model improves [28]. Their model achieved a top-20 retrieval accuracy of 81.9 on the Natural Questions (NQ) dataset, outperforming the 78.4 achieved by DPR. Despite these improvements, ANCE introduces significant computational overhead. Refreshing the global index for a large corpus is computationally expensive as it requires a forward pass on all the documents. If the refresh rate is too low, it fails to solve the stale negative problem, leading to training fluctuations. Additionally, ANCE does not resolve the false negative problem as it lacks a mechanism for denoising hard negatives.

RocketQA (2021) addressed these remaining challenges by introducing three optimized training strategies: cross-batch negatives, denoised hard negatives, and data augmentation. To resolve the false negative problem, RocketQA trains a cross-encoder to identify and remove unlabeled positives from the hard negatives set. It also uses cross-batch negatives to share vector embeddings across multiple GPUs, which increases the number of negatives available for each query. Lastly, the authors use a cross-encoder to generate pseudo-labels for unlabeled questions, thereby increasing the available training data. Their method achieved an MRR of 37.0 on the MS MARCO passage ranking task, outperforming the 33.0 achieved by ANCE. On the Natural Questions (NQ) dataset, RocketQA reached a top-20 retrieval accuracy of 82.7, higher than the 81.9 score of ANCE. While RocketQA successfully addresses the false negative problem, it does so by introducing a highly complex and expensive training pipeline. The process requires multiple stages, including training a cross-encoder specifically for denoising negatives and generating pseudo-labels before the dual-encoder can even be trained. Furthermore, the cross-batch strategy relies on distributed hardware to manage vector sharing across multiple GPUs, making the initial training just as resource-intensive as ANCE.

Representational pre-training

Other methods, such as coCondenser and SimLM, focus on improving representation learning during pre-training to address the [CLS] token bottleneck [15]. Since the [CLS] vector often fails to capture the full semantic content of a document, these approaches re-engineer the pre-training architecture to produce more informative representations. This is done by forcing the model to compress essential information into a single vector before fine-tuning begins.

The Condenser architecture adds an auxiliary head during pre-training that combines the final-layer [CLS] vector with a “short-circuit” connection from earlier layers. This differs from standard models like BERT, which rely solely on the final layer and aggregate information from all neighboring tokens to predict masked text. By including signals from the earlier layers, Condenser helps the [CLS] token capture a richer and more meaningful representation of the whole document. [8].

Similarly, SimLM uses a deep encoder and a shallow decoder that are connected solely through the [CLS] vector. During pre-training, the model replaces masked tokens with words generated by an ELECTRA-style generator. Because the decoder has only two layers, it is almost completely dependent on the [CLS] vector to successfully predict the original tokens. This design encourages the encoder to compress essential semantic information into a single dense vector, resulting in more informative representations for retrieval tasks [19]. While these models still require fine-tuning, they achieve similar performance with significantly smaller training batches, such as coCondenser reducing the required batch size from 4096 to 64, and similarly SimLM.

Additional Negative Sampling Techniques

Forms of more ”lightweight” negative mining strategies have also been explored in research. Zhan et al. use a combination of STAR and ADORE in order to effectively and efficiently train dense retrieval models. STAR uses a pre-trained model to retrieve the top documents for all training queries just once. These serve as the static hard negatives. It also selects random negatives by reusing document embeddings from other queries in the same batch. This combination ensures a steady training process while also improving the retrieval of top-ranked results. ADORE uses a pre-trained document encoder to build a fixed index once, then re-encodes queries at every training step to retrieve ”real” dynamic hard negatives that directly optimize ranking performance. To achieve best-in-class performance, the authors use STAR to train the document encoder and then apply ADORE to further train the query encoder [29].

TAS-B looks to replace continuous index refreshing and massive batch sizes by using a topic-aware sampling strategy. Essentially, it groups queries into clusters so that each batch contains only related queries from a single topic. As a result, the sampled negatives are semantically related and harder to distinguish from relevant results. Additionally, within each group, the authors use knowledge distillation tools to categorize document pairs into “bins” according to their difficulty, and uniformly sample between these bins to ensure a consistently challenging learning signal [12].

ICT-Q and ICT-P build on top of TAS-B and ANCE by replacing the teacher model with representations

generated by the model itself to perform clustering. Additionally, they also re-use the idea of re-indexing suggested by ANCE by periodically refreshing or updating the clustering representations during training [21]

Prioritizing Training Examples

The methods above select negatives that a retriever or teacher scores as similar to the query. Other methods use the model's own loss or error to decide how often each training example is used, and update these priorities as the model learns.

Online Batch Selection (OBS) looks to speed up training by changing which examples form each batch. Optimizers such as AdaDelta and Adam visit every example equally often, even though some examples have a much higher loss than others. OBS instead ranks all examples by their latest known loss and samples them with a probability that decreases exponentially with their rank, so high-loss examples are selected more often. On MNIST, this made both optimizers about five times faster [16].

Prioritized Experience Replay (PER) applies a similar idea to reinforcement learning. Deep Q-Networks (DQN) replay past experiences uniformly, regardless of how much the agent can still learn from them. PER instead ranks experiences by their temporal-difference error, which measures how far the agent's prediction is from its target, and samples them with a probability that decreases as a power of their rank. This improved DQN in 41 of 49 Atari games, raising the median normalized score from 48% to 106% [23].

Reducible Holdout Loss Selection (RHO-LOSS) shows that selecting by loss, as OBS and PER do, often picks mislabeled, ambiguous, or outlier examples. When labels were randomly corrupted, loss-based selection picked far more mislabeled examples than uniform sampling. RHO-LOSS instead trains only on the 10% of each large batch with the highest score: the current loss minus the loss of a small model trained on held-out data. Mislabeled and already learned examples therefore score low. On the noisy Clothing-1M dataset, it needed 18 times fewer steps than uniform sampling and reached a 2% higher final accuracy [17].

SimANS (Simple Ambiguous Negatives Sampling) brings score-based sampling to the negatives of each query in dense retrieval. It samples each query's top-ranked candidates with a probability that peaks near the positive's score rather than at the highest scores. Two hyperparameters set the position and width of this peak. When applied to ANCE, it improved the top-20 retrieval accuracy on NQ from 81.9 to 83.0 [30].

These methods show that a model's own training signal can guide which examples it trains on, but that the hardest examples are not always the most useful. None of the dense retrieval methods above applies this to the training queries themselves.

Uncertainty-Guided Selection

The methods above rank negatives and training examples by a single score, such as a similarity or a loss. Such a point estimate does not indicate how certain the model is about each candidate, so two candidates with the same score are treated alike, even if the model is far more certain about one. Uncertainty estimation adds this information and can guide which documents are ranked or which negatives are trained on.

Monte Carlo (MC) dropout estimates this uncertainty without changing the network. Bayesian neural networks express uncertainty but are expensive to train. Training with dropout, which randomly switches off parts of a network, already approximates one. Keeping dropout active at test time and passing the same input through it several times therefore gives a spread of outputs, whose mean is the prediction and whose variance is the uncertainty [7]. In neural ranking, dropout in two small layers added on top of the ranker gives each document a distribution of relevance scores at almost no extra cost. Ranking documents by the upper or lower end of this distribution raised nDCG@200 on TREC DL 2019 to about 0.61, compared with 0.582 for the same ranker without dropout [7].

UnReMix (Uncertainty and Representativeness Mixing) uses uncertainty to weight negatives in self-supervised contrastive learning. Each anchor is contrasted against the other inputs in its batch. Since

no labels are available, some of these negatives belong to the anchor's class. Weighting negatives by similarity alone gives these more weight in the loss. UnReMix instead combines this similarity with the model's uncertainty about each negative and its average distance from the other negatives. According to the authors, same-class negatives are close to the anchor, but the model is confident about them, so their low uncertainty lowers their weight. The distance term keeps the negatives diverse. On CIFAR-10, weighting by uncertainty alone instead of similarity alone improved accuracy from 91.19% to 91.88%, and weighting by distance alone improved it to 92.02%. However, it was not evaluated on passage retrieval [24].

These methods show that a model's uncertainty can be estimated without training a separate model, and that weighting negatives by uncertainty can work better than weighting them by similarity alone. None of them use this uncertainty to select hard negatives for a dense retriever, which this thesis examines.

Systems Optimization for low hardware resources

A larger pool of negatives improves contrastive training [11]. In standard in-batch training, however, every negative must be encoded within the batch and kept in memory for backpropagation.

Gradient Cache (GradCache) decouples backpropagation between the contrastive loss and the encoder, which removes the dependency of the encoder's backward pass on the rest of the batch. This enables large-batch gradient updates to be computed in memory-limited settings while producing the exact same gradients as full-batch training [9]. It still encodes every negative at every step, so the negatives remain limited to the batch.

Another approach stores embeddings computed at earlier training steps and reuses them as negatives. These embeddings are stale, because the encoder has changed since they were computed. Momentum Contrast (MoCo) keeps the embeddings of recent batches in a queue as negatives for self-supervised image representation learning. The queued embeddings come from different training steps, so MoCo computes them with a separate encoder that follows the trained encoder as a slowly moving average. Without this average, training failed to converge [11].

Cross-Batch Memory (XBM) shows that stale embeddings can still be used for hard-negative mining. In deep metric learning for image retrieval, hard negatives are usually mined within a single batch, which contains few informative pairs. XBM measured how far the embedding of the same image moves during training. The change was large early in training and then became small. After a short warm-up, XBM therefore stores the embeddings of recent batches and compares each anchor with the whole memory. Unlike in MoCo, these embeddings come from the trained encoder itself. The authors bound the resulting gradient error by how far the stored embeddings have drifted. The memory supplied about 1,000 valid hard negatives per iteration, compared to fewer than 10 from the batch alone. With a batch of 64 and a contrastive loss, storing the whole training set raised Recall@1 on Stanford Online Products from 63.9% to 77.4% for 0.2 GB of additional GPU memory. A batch of 256 without the memory reached 71.7% and needed 15.7 GB more [27].

Negative Cache uses a similar idea for dual-encoder retrieval, where the document encoder must also be trained. It keeps a cache of document embeddings. Each query is scored against all cached documents. One negative is sampled from the resulting softmax distribution using the Gumbel-Max trick. This negative and the positive are then encoded again with the current model, so the loss also trains the document encoder. After each step, the oldest cached embeddings are recomputed. A streaming variant stores only a random subset of the corpus and replaces its oldest entries with newly sampled documents. The authors prove that refreshing only a small fraction of the cache per step does not change the convergence rate of stochastic gradient descent. On MS MARCO, a cache of two million passages reached an MRR@10 of 0.331, compared to 0.330 for ANCE. On Natural Questions, it trailed ANCE in Recall@20, at 0.784 against 0.819 [14].

These methods show that negatives can be drawn from stored embeddings that are not recomputed at every step, provided part of the store is kept up to date. They refresh or replace stored entries by age or at random. None of them decides which entries to keep, refresh, or replace based on how useful they are as hard negatives.

Summary

This chapter reviewed work related to one problem: obtaining informative negatives for a dense re-retriever at an acceptable computational cost. The review covered negative mining and representation pre-training for dense retrieval, the prioritization of training examples, uncertainty estimation, and training under limited GPU memory.

Dense retrieval enables semantic search over large corpora, but its effectiveness depends on the negatives used during training. DPR's negatives were often easy, and some were false negatives. ANCE and RocketQA therefore mine hard negatives from the full corpus with a dense retriever. This improved retrieval accuracy, but required repeated re-encoding of the corpus, or a multi-stage pipeline with a cross-encoder and large batches spread across multiple GPUs.

Later methods reduce parts of this cost. coCondenser and SimLM strengthen the encoder during pre-training, which allows fine-tuning with much smaller batches. STAR and ADORE retrieve negatives from a document index that is built only once. TAS-B and ICT compose harder batches by clustering queries or passages. Table 2.1 summarizes which model mines the negatives of each method, where they come from, and how often they are updated.

All of these methods select negatives that a retriever or teacher scores as similar to the query. Example prioritization methods instead rely on the model's own training signal. OBS, PER, and RHO-LOSS use the loss or error of each training example to decide how often it is used. SimANS brings this kind of score-based sampling to the negatives of a dense retriever. It still scores them by similarity, but favors those near the positive's score over the highest-scoring ones. RHO-LOSS and SimANS also show that the hardest examples are not always the most useful, as they include mislabeled examples or false negatives. None of the reviewed dense retrieval methods apply such selection to the training queries (Table 2.2).

Similarity and loss are both point estimates and do not indicate how certain the model is about a score. MC dropout estimates this uncertainty from the trained network itself. Uncertainty has been used to re-rank documents and, in UnReMix, to reduce the weight of likely same-class negatives in contrastive learning. None of these methods uses it to select hard negatives for a dense retriever (Table 2.3).

A larger pool of negatives improves contrastive training, but GPU memory limits the number of negatives in a batch. GradCache decouples the batch size from GPU memory but still encodes every negative at every step. MoCo, XBM, and Negative Cache instead reuse embeddings computed at earlier training steps. These embeddings are stale, but remain usable when the encoder changes slowly, or part of the stored embeddings is refreshed. All three refresh or replace stored embeddings by age or at random, rather than by their usefulness as hard negatives (Table 2.4).

These three gaps, in query selection, uncertainty-guided negative selection, and the maintenance of stored negatives, motivate the method described in the Methodology chapter.

Table 2.1: Overview of negative sampling in dense retrieval methods.

Model	Source	Mining Model	Hard Negative Source	Iterative Updates
DPR (In-batch)	[22]	None	Batch (Random)	None
DPR (BM25)	[22]	BM25 (Static)	Full Corpus	None
ANCE	[28]	Self (Dense)	Full Corpus	Async
RocketQA	[20]	Self (Dual-Enc)	Full Corpus (Denoised)	Multi-stage
coCondenser	[8]	Self (Retriever)	Full Corpus	2-round
SimLM	[19]	Self + Re-ranker	Full Corpus	3-stage
STAR	[29]	Warm-up (Fixed DR)	Full Corpus (Static)	None
ADORE	[29]	Self (Query-only)	Full Corpus (Dynamic)	Step-wise
TAS-B	[12]	Teacher (Clustering)	Training Queries	None
ICT-P	[21]	Self (Clustered)	Training Passages	Periodic
ICT-Q	[21]	Self (Clustered)	Training Queries	Periodic
SimANS	[30]	Base DR	Top-k (Near-positive)	As base method
Negative Cache	[14]	Self (Gumbel-Max)	Global Cache	Step-wise

Table 2.2: Overview of the example-selection, uncertainty, and memory methods that inform this thesis.

Method	Setting	Applies to	Based on
Prioritizing training examples			
OBS [16]	Image classification	Batch examples	Loss rank (exponential)
PER [23]	Reinforcement learning	Agent steps	TD-error rank (power)
RHO-LOSS [17]	Image classification	Batch examples	Loss minus reference loss
Uncertainty			
MC dropout [7]	Any dropout network	Predictions	Variance over dropout passes
MC dropout ranking [5]	Neural re-ranking	Document scores	Tail of the score distribution
UnReMix [24]	Contrastive learning	In-batch negatives	Similarity, gradient-based uncertainty, distance
Training under limited GPU memory			
GradCache [9]	Dense retrieval	Batch negatives	Gradient caching
MoCo [11]	Contrastive learning	Stored embeddings	Momentum encoder, replaced by age
XBM [27]	Image retrieval	Stored embeddings	Trained encoder, replaced by age

3

Methodology

3.1. Data, Preprocessing, and Evaluation

This section describes the datasets, preprocessing, and evaluation procedures used in the experiments. The individual training and negative selection algorithms are presented in the subsequent sections.

3.1.1. Base Models

BGE-M3 Configuration for BRIGHT

All dense retrievers in the main BRIGHT comparison use BAAI/bge-m3. BGE-M3 is an XLM-RoBERTa-based model with 567,754,752 parameters that supports sequence lengths of up to 8192 tokens and produces 1024-dimensional embeddings. It is pretrained and fine-tuned on multilingual and multi-domain data, including MS MARCO, Natural Questions (NQ), HotpotQA, MIRACL, Mr. TyDi, and MLDR [4]. All dense retrievers in the main BRIGHT comparison use BAAI/bge-m3. BGE-M3 is an XLM-RoBERTa-based model with 567,754,752 parameters that supports sequence lengths of up to 8192 tokens and produces 1024-dimensional embeddings. It is pretrained and fine-tuned on multilingual and multi-domain data, including MS MARCO, Natural Questions (NQ), HotpotQA, MIRACL, Mr. TyDi, and MLDR [4].

Queries and passages are tokenized independently. Queries are truncated to 1,024 tokens and passages are encoded as described in Section 3.1.5. Section 3.1.6 reports how much of the training and evaluation data is retained under these limits. BGE-M3 encodes each sequence into a 1024-dimensional dense vector using `[CLS]` pooling. The resulting embeddings are L2-normalized, making inner-product similarity equivalent to cosine similarity.

During training, similarity scores are divided by a temperature of $\tau = 0.02$ before the contrastive loss is computed.

Table 3.1 summarizes the shared configuration for the dense retrievers.

Table 3.1: Shared dense-encoder configuration.

Property	Configuration
Base model	BAAI/bge-m3
Representation	Dense retrieval mode
Pooling	CLS pooling
Embedding normalization	L2 Normalization
Similarity function	Normalized dot product
Maximum query length	1024 tokens
Segment length	512 tokens
Segments per passage	Up to 4, best segment (MaxP)
Training temperature	0.02
Training precision	Float32

RoBERTa Configuration for the ANCE Validation

The separate MS MARCO ANCE validation follows the original ANCE Passage setup rather than BGE-M3. Queries and passages are encoded by the same RoBERTa-base model, followed by a linear projection and LayerNorm that produce 768-dimensional embeddings. Similarity is computed using raw inner product. Training used mixed precision: model parameters are stored in float32, while query and passage encoding and loss computation use bfloat16 where supported.

Before ANCE training, RoBERTa-base is fine-tuned for 60,000 optimization steps using precomputed BM25 negatives. Queries and passages are limited to 128 tokens during this stage. The resulting checkpoint initializes the subsequent ANCE training stage.

3.1.2. Datasets

BRIGHT

BRIGHT is a challenging text retrieval benchmark designed to evaluate domain understanding and reasoning. Standard lexical retrievers and general-purpose bi-encoders perform poorly on many of its tasks because relevant documents may use different wording from the query. Because these retrieval tasks require domain knowledge and reasoning, effective models must distinguish relevant passages from plausible but incorrect alternatives. BRIGHT is therefore suitable for comparing the training strategies and evaluating whether uncertainty-guided negative selection improves this ability.

The benchmark contains 1384 queries across 12 domains, with a separate document corpus for each domain. Each query is associated with one or more relevant document identifiers. These query-document pairs are assigned a binary relevance label of 1, while documents without a positive label are treated as non-relevant during evaluation. The dataset also provides query-specific document IDs that must be excluded from retrieved rankings before scoring. This reduces false negatives by preventing potentially relevant but unlabeled documents from being treated as non-relevant. The implementation loads the BRIGHT `documents` and `examples` configuration for the domain corpora and evaluation data, respectively.

The domains cover scientific, social-science, mathematical, and programming tasks. Table 3.2 reports the number of queries and documents evaluated in each domain.

Table 3.2: Query and document counts for the BRIGHT evaluation domains.

Domain	Queries	Documents
Biology	103	57,359
Earth science	116	121,249
Economics	103	50,220
Psychology	101	52,835
Robotics	101	61,961
Stack Overflow	117	107,081
Sustainable living	108	60,792
Pony	112	7,894
LeetCode	142	413,932
AoPS	111	188,002
TheoremQA theorems	76	23,839
TheoremQA questions	194	188,002
Total queries	**1,384**	

ReasonIR

The ReasonIR dataset is used to train the models on reasoning-intensive tasks. It consists of synthetic retrieval queries and passages generated using Llama3.1-70B-Instruct and is divided into varied-length (VL) and hard-query (HQ) components. The authors used this dataset to train a Llama-based ReasonIR-8 B, which is not used in this thesis.

For VL, Llama3.1-70B is prompted to propose long-document retrieval tasks that describe the type of query and positive passages to generate. For example, one task is to generate a long business

proposal as the query and a passage describing relevant competitive strategies as the positive. The model receives this instruction together with a target length between 300 and 2000 words. The prompt also instructs the model to avoid substantial word overlap between the query and positive passages. In a separate generation step, the model receives the query and positive passages and generates a hard negative that appears relevant but does not contain the information needed to address the query.

For HQ, existing documents from the BRIGHT corpus are passed to Llama3.1-70B-Instruct with instructions to generate difficult, self-contained queries for which the documents are relevant. Unlike VL, the queries and positive passages are not generated together using target lengths. Instead, the selected BRIGHT documents serve as the positive passages. Each query and passage is then used to generate a hard negative following the same procedures as VL.

MS MARCO

MS MARCO is a widely used open-domain passage-retrieval benchmark containing over 500,000 anonymized search queries from Microsoft Bing logs and 8,841,823 passages extracted from web documents [2].

This dataset is used in two distinct experiments. In the primary experiment, a subset of 83,030 MS MARCO triplets is combined with ReasonIR to form the shared training dataset used by all BGE-M3 training strategies. The resulting retrievers are evaluated on BRIGHT to compare their retrieval performance and training efficiency.

In a separate experiment, the complete MS MARCO passage corpus is used to validate the ANCE implementation against the performance reported in the original study. The implementation follows the original experimental setting where possible and is evaluated on the MS MARCO passage-ranking development set (MS MARCO Dev).

Table 3.3 summarizes the data and evaluation benchmark used in each experiment.

Table 3.3: The two uses of MS MARCO in this thesis.

Experiment	Training data	Evaluation data	Purpose
Retrievers evaluated on BRIGHT	83,030 sampled MS MARCO-derived triplets	BRIGHT domains	Provide general passage-retrieval supervision within the shared training mixture
Full-corpus ANCE validation	8,841,823 passages and 400,782 usable training records	6,980 MS MARCO development queries with 7,437 relevance judgments	Validate the ANCE implementation at the corpus scale and retrieval setting used in the original study

For the BRIGHT experiment, the MS MARCO examples are obtained from the `triplet` configuration of `sentence-transformers/msmarco-distilbert-margin-mse-mean-dot-v1`. Each triplet contains a query, a relevant passage, and a mined negative passage.

For the full-corpus validation, all 8,841,823 passages are indexed and used as the search collection. The processed training data contain 400,782 queries and 426,215 known relevant query–passage pairs because some queries have more than one relevant passage. Each training record also provides between one and thirty precomputed BM25 negatives. Pairing each query and its first listed positive passage with the supplied negatives produces 11,920,727 available triplets, or approximately 29.7 triplets per query.

3.1.3. Training-Data Composition and Preprocessing

The BGE-M3 retrievers evaluated on BRIGHT use a shared mixture of MS MARCO triplets, ReasonIR VL, and ReasonIR HQ. The final mixture contains 329,993 records. Table 3.4 summarizes the dataset composition.

After the three component files are generated, the complete mixture is validated before the mining files are created. Validation rejects missing or empty files, records missing a query identifier, query text, positive passages, and negative passages, duplicate query identifiers, and cases where the same document identifier refers to different passage texts. Seven VL records fail passage checks, reducing the requested total of 330,000 records to 329,993.

Table 3.4: Composition and preprocessing of the common training dataset.

Component	Final records	Selection, validation, and purpose
MS MARCO	83,030	Fixed-seed sampling. Blank records are rejected, and equal positive and negative passages are skipped. General passage-retrieval supervision.
ReasonIR VL	149,963	The first 95,000 rows are excluded because this part of the dataset is dominated by label-style examples with very short positive passages. Records without valid positive and negative passages are rejected. Length generalization.
ReasonIR HQ	97,000	Positive document IDs are resolved through the BRIGHT corpora. Unusable records are rejected. Reasoning-intensive supervision.
Total	**329,993**	

Original training records remain unchanged. Their positive and negative passages are collected into a separate mining corpus, where passages with identical text are stored only once. This produces 648,942 unique passages. When identical text occurs with multiple document IDs, an ID associated with a positive passage is retained. The training relevance file is updated to use this ID, allowing the mining algorithms to recognize the passage as relevant and exclude it from negative selection.

Relationship to BRIGHT evaluation data As described above, the ReasonIR HQ positives are drawn from the BRIGHT corpora. An exact text comparison found no identical queries between the 329,993 training records and the 1,384 BRIGHT evaluation queries. However, 3,234 of the 4,712 unique documents marked as relevant for BRIGHT evaluation also occur as positive passages for different HQ queries.

3.1.4. Data Formatting and Artifacts

The BRIGHT, ReasonIR, and MS MARCO datasets are first converted into the JSONL formats compatible with Tevatron. Each training record contains a query together with its positive and negative passages, while separate corpus and query files are used for encoding. For BM25, the processed BRIGHT corpus is converted to Pyserini format [13], while the same query files are used directly by both sparse and dense retrieval.

Table 3.5: Shared data formats and their consumers.

Artifact	Format	Purpose and consumers
Training mixture	Tevatron JSONL	Contains a query and its positive and negative passages. Used by all training strategies.
Mining corpus	JSONL (`docid`, `text`)	Contains the unique passages encoded and searched during hard-negative mining.
Mining queries	JSONL (`query_id`, `query`)	Contains the training queries encoded during hard-negative mining.
Training qrels	Four-column TREC	Lists the positive documents for each training query so they can be excluded from negative selection.
BRIGHT corpus	JSONL (`docid`, `text`)	Used directly for dense encoding and converted to Pyserini format for BM25 indexing.
BRIGHT queries	JSONL (`query_id`, `query`)	Provides the same queries to dense retrieval and BM25.
BRIGHT qrels	Four-column TREC	Lists the relevant documents for each query and is used to compute evaluation metrics.
BRIGHT exclusions	JSON	Lists the documents that must be removed from the ranking for each query.

Query and document IDs remain consistent across related files so that queries, passages, relevance

labels, and exclusions can be matched correctly. Preprocessing rejects missing or empty files, duplicate query IDs, and conflicting document IDs associated with different passage texts.

3.1.5. Document Segmentation (MaxP)

Each passage and corpus document is split into at most four consecutive, non-overlapping 512-token segments, and each segment is encoded separately. A query is compared with every segment of a document, and the document receives the highest of these scores:

$$s(q, d) = \max_k \mathbf{q}^\top \mathbf{d}_k, \tag{3.1}$$

where $\mathbf{q}$ is the query embedding and $\mathbf{d}_k$ is the embedding of segment k. The best segment is selected separately for each query, so different queries can match different parts of the same document. Segments are limited to 512 tokens because two training epochs at a 1,024-token limit would take beyond 24 hours to train.

This score is used wherever a query is compared with a passage or document, including the contrastive loss (Equation 3.2), hard-negative mining, and evaluation.

3.1.6. Token Limits and Coverage

Queries are truncated after 1,024 tokens, which fully retains 99.75% of the 329,993 training queries and 98.12% of the 1,384 BRIGHT evaluation queries. Training passages are mostly short. Of the 648,942 unique passages in the mining corpus, 91.16% fit in a single segment, and only 0.11% of the training positives exceed four segments. Relevant BRIGHT documents are much longer. Only 59.61% fit in a single segment, so the remaining 40.39% rely on additional segments 3.6

Table 3.6: BRIGHT query retention within 1,024 tokens and relevant-document coverage by one and by four 512-token segments.

Domain	Queries fully retained	Relevant documents within one segment	Aggregate relevant tokens in the first segment	Aggregate relevant tokens within four segments
Biology	100.00%	58.60%	50.26%	86.54%
Earth science	100.00%	39.83%	18.42%	41.70%
Economics	100.00%	62.00%	76.00%	99.11%
Psychology	100.00%	72.67%	64.55%	93.05%
Robotics	81.19%	67.50%	86.93%	100.00%
Stack Overflow	96.58%	8.42%	46.18%	100.00%
LeetCode	97.89%	37.96%	75.13%	100.00%
AoPS	100.00%	72.97%	83.12%	100.00%
TheoremQA theorems	100.00%	63.16%	72.49%	94.60%
TheoremQA questions	100.00%	87.88%	93.76%	100.00%
Sustainable living	100.00%	87.15%	68.45%	84.47%
Pony	100.00%	100.00%	100.00%	100.00%
Overall	**98.12%**	**59.61%**	**47.67%**	**75.33%**

Stack Overflow, LeetCode, and Earth Science rely most on the additional segments, with only 8.42%, 37.96%, and 39.83% of their relevant documents fitting in one segment. Across all domains, four segments raise the share of relevant tokens retained from 47.67% to 75.33%. Only 3.69% of the relevant documents exceed four segments, mostly in Earth Science (22.05%) and Biology (7.26%)

3.1.7. Evaluation Pipeline

Each retriever is evaluated separately on the 12 Bright domains using the same preprocessed corpora, queries, relevance judgments, and exclusion lists. Dense retrieval scores each document by its highest-

scoring segment (Equation 3.1, while BM25 uses term-matching scores. The base BGE-M3 model and the fine-tuned models use the same document aggregation. Both methods then follow the same filtering and evaluation procedure shown in Figure 3.1.

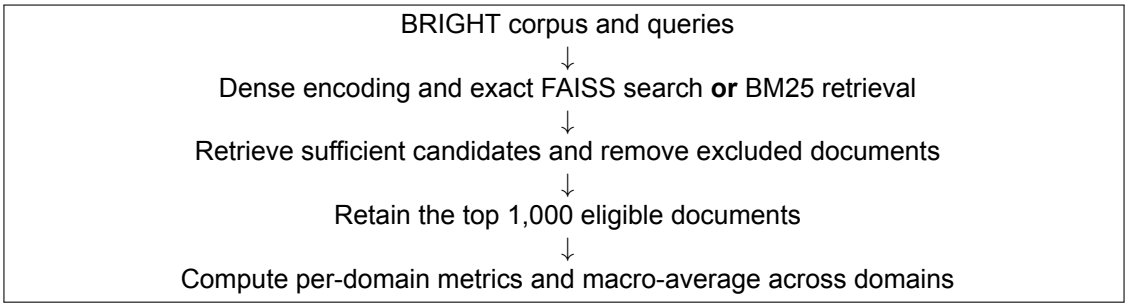

Figure 3.1: Shared BRIGHT evaluation procedure.

Retrieval Configuration and Exclusion Filtering

Dense retrieval encodes each domain's corpus and queries using Tevatron and searches the resulting embeddings with an exact FAISS `IndexFlatIP` index [10, 6]. Because the embeddings are L2-normalized, inner-product ranking is equivalent to cosine-similarity ranking, as described in Table 3.1. BM25 indexes the same corpus text in Lucene and performs retrieval through Pyserini [13].

The BM25 parameters $k_1 = 0.9$ and $b = 0.4$ are taken from the BRIGHT paper [3] and used without further tuning. Table 3.7 summarizes these settings.

For each query, the retriever returns at least 1,000 candidates plus enough additional documents to account for its exclusions. The excluded document IDs are then removed, and the remaining ranking is truncated to the 1,000 highest-scoring documents. These documents are then compared with the query's known relevant document IDs during evaluation.

Table 3.7: Retrieval configuration used for BRIGHT evaluation.

Property	Dense retrieval	BM25
Implementation	Exact FAISS `IndexFlatIP`	Pyserini and Lucene
Ranking function	Highest normalized inner product over segments	BM25 with $k_1 = 0.9$ and $b = 0.4$
Corpus and queries	Per-domain BRIGHT data	Same corpus and queries
Initial retrieval depth	$1{,}000$ plus the current query's exclusion count	$1{,}000$ plus the current query's exclusion count
Final retrieval depth	1,000 eligible documents	1,000 eligible documents
Query expansion	None	None
Reranking	None	None

Metrics, Aggregation, and Result Validity

The filtered rankings are evaluated using `pytrec_eval` and the binary relevance judgements provided by BRIGHT [26]. Recall@1000 is the primary metric, while NDCG@10 and MRR provide additional information about the positions of relevant documents. Table 3.8 summarizes these metrics.

Table 3.8: Metrics reported for BRIGHT evaluation.

Metric	Role	Interpretation
Recall@1000	Primary	Fraction of relevant documents retrieved in the first 1,000 results
NDCG@10	Secondary	Discounted ranking quality over the first ten results
MRR	Secondary	Reciprocal rank of the first relevant document

Metrics are calculated separately for each domain and averaged over the queries listed in its relevance judgments. A listed query missing from the retrieval results receives a score of zero. A retrieved query not listed in the relevance judgments is excluded because no relevant documents are available for

evaluating it. The final score is calculated by averaging the 12 domain scores. Recall@1000 is the primary metric, while NDCG@10 and MRR are reported as secondary metrics.

Differences between two retrievers are tested with a two-sided paired t-test over the per-query Recall@1000 differences. Each domain contributes equally to the test, regardless of its number of queries. A difference is considered significant at $p < 0.05$. The three GRASS comparisons test difficulty-based against uniform query replay. Each model is trained once, so the test does not capture variation between training runs due to the high training times.

Several validation checks are performed before the final scores are reported. For dense evaluation, the query file, exclusion list, and encoded outputs must contain identical sets of query IDs. Every query with a relevance judgement must also occur in the query file. The final aggregate score is reported only after all 12 domains complete successfully, and all combined domain scores must originate from the same model checkpoint.

MS MARCO ANCE Evaluation

The separate ANCE implementation is evaluated on the MS MARCO Dev using 6,980 queries and the complete corpus of the 8,841,823 passages. The trained RoBERTa-based model encodes the queries and passages, after which exact retrieval is performed with a FAISS `IndexFlatIP` index using raw inner-product similarity. The top 1,000 passages are retained for each query. MRR@10 is calculated from the first ten results, while Recall@1000 is calculated from the top 1,000 results. The scores are averaged over all evaluation queries and compared with MRR@10 of 0.330 and Recall@1000 of 0.959 reported for the 600,000-step ANCE checkpoint [28].

3.2. Training and Negative-Selection Strategies

This section describes the training and negative-selection strategies used in the experiments. The BGE-M3 experiments use the shared contrastive objective described below, but differ in how their negative candidates are constructed. Cross-batch training also changes the effective batch size and number of optimizer updates. GRASS additionally chooses which queries are used for training and uses a smaller training budget, described in Section 3.2.6. The separate MS MARCO ANCE validation uses a pairwise training objective. The section concludes by introducing GRASS, the uncertainty-guided hard-negative selection method proposed in this thesis.

3.2.1. Contrastive Loss for the BRIGHT Experiments

The retrievers use a softmax cross-entropy contrastive loss that increases the similarity between each query and its positive passages relative to its negative passages. Given a query q_i, its positive passage p_i^+, and n negative passages $p_{i,1}^-, \ldots, p_{i,n}^-$, the negative log-likelihood objective used in dense retrieval [22] is

$$\mathcal{L}\left(q_i, p_i^+, p_{i,1}^-, \ldots, p_{i,n}^-\right) = -\log \frac{\exp\left(s(q_i, p_i^+)/\tau\right)}{\exp\left(s(q_i, p_i^+)/\tau\right) + \sum_{j=1}^{n} \exp\left(s(q_i, p_{i,j}^-)/\tau\right)}. \tag{3.2}$$

The loss is averaged over the queries in each training step, with the negative passages determined by the training strategy.

3.2.2. Batch Construction

The in-batch, cross-batch, and ANCE strategies train for two epochs. GRASS uses a smaller budget, described in Section 3.2.6.

All BGE-M3 training strategies shuffle the examples within each source of the training mixture and divide them into batches. Within these batches, each query's in-batch negatives come from the same source as the query. Examples left over after the last full batch of each source are combined into additional batches that may mix sources, and any final incomplete batch is dropped. The completed batches are then processed in random order.

3.2.3. In-Batch Negatives

The in-batch baseline enlarges the negative pool for each query by reusing passages from other examples in the same batch. Each query is paired with one positive and one predefined hard negative, and is trained against these and the positive and hard-negative passages of all other queries in the batch.

Batches are re-formed within each source at every epoch, so the passages contributed by the other examples in a batch can change. However, each query retains the same predefined hard negative. Known positives of the query are filtered out of its negatives.

3.2.4. Cross-Batch Negatives

Cross-batch training enlarges the negative pool beyond standard in-batch training by processing local batches on multiple GPUs in parallel. These local batches are formed by drawing one batch from a single source (Section 3.2.2) and splitting it evenly across the GPUs. Each GPU's candidate pool is shared across the GPUs and combined into a single larger pool. The contrastive loss then compares each query with all passages in this combined pool [9].

Batch size is limited by GPU memory because the encoder activations for all queries and passages must normally be retained for backpropagation. GradCache makes larger contrastive batches possible by processing the inputs in smaller chunks [9]. It first encodes the chunks without retaining their intermediate activations and stores the resulting embeddings. The contrastive loss is then computed over the complete set of embeddings, and the loss gradients for these embeddings are stored. Each chunk is subsequently re-encoded to backpropagate its stored gradients through the model. The resulting parameter gradients are accumulated before a single optimizer update is applied. This produces the same update as direct large-batch training while retaining encoder activations for only one chunk at a time. The additional encoding pass reduces memory use at the cost of increased training time. Gao et al. provide the formal derivation [9].

The 1,024 queries of one update form one batch of the source-based batching (Section 4.3.1) and are divided between the two GPUs.

3.2.5. Asynchronous ANCE training.

ANCE refreshes hard negatives during training by periodically re-encoding the corpus with a recent checkpoint of the model being trained and searching it for each query [28]. The shared mining and refresh cycle is described first, followed by its adaptation for BRIGHT and the differences required for the MS MARCO validation.

Mining and Refresh Cycle

During asynchronous training, one A100 GPU is dedicated to model training and a second A100 GPU to hard-negative mining. The training GPU continues updating the model using the current negatives while the mining GPU prepares their replacements from a recently saved checkpoint.

Before training begins, the initial model encodes the complete mining corpus and all training queries into dense vectors. The corpus vectors are indexed using FAISS `IndexFlatIP`, which performs an exhaustive inner-product search [6]. Each training query retrieves its 200 highest-scoring documents from the indexed corpus. Known positives are removed, and one negative is sampled uniformly from the remaining candidates. This document replaces the predefined negative in the corresponding training record.

The negatives mined by the initial model are used to start training. During training, GPU 0 periodically saves checkpoints every 1,000 optimizer updates. After GPU 1 finishes its current mining round, it polls for new checkpoints every 60 seconds. If a newer complete checkpoint is found, it loads the most recent one, re-encodes the complete corpus and all training queries, rebuilds the FAISS index, and mines a new set of negatives for each query. GPU 0 continues training with the previous negatives throughout this process.

GPU 0 checks for completed negative sets every 100 optimizer updates. While training, once a new set is detected, the trainer replaces the previous training records with the newly mined ones and continues training with the refreshed negatives.

Figure 3.2 summarizes the concurrent procedure.

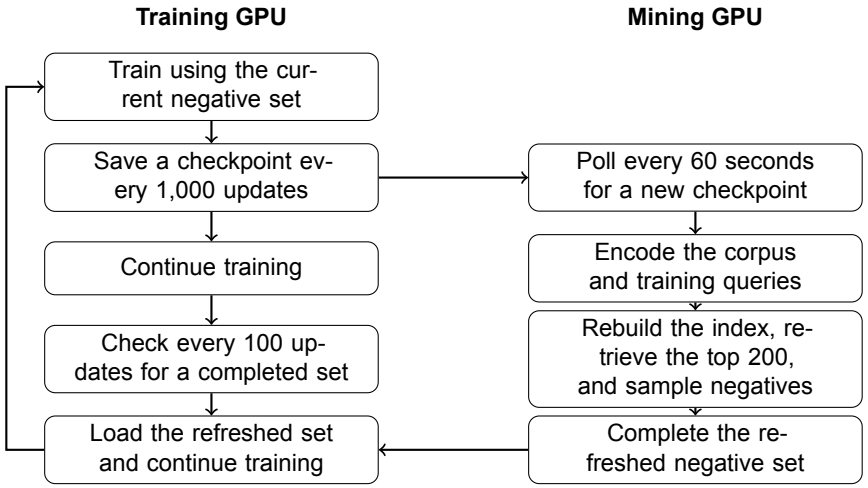

Figure 3.2: Concurrent training and hard-negative refresh in the BRIGHT ANCE-style experiment.

ANCE-Style Mining for BRIGHT

The BRIGHT experiment starts from the pretrained BGE-M3 model and uses the contrastive objective described earlier. As in the in-batch baseline, each query is also compared with the passages of the other queries in its batch.

This differs from the original ANCE implementation, which uses LAMB and a pairwise objective. The purpose of the BRIGHT experiment is not to reproduce the original ANCE training setup, but to compare hard-negative selection methods while keeping the remaining bGE-M3 training settings consistent.

MS MARCO ANCE Implementation Validation

The MS MARCO validation uses the original pairwise ANCE objective and LAMB optimizer. Because the released 60,000-update warm-up checkpoint is unavailable, it is recreated locally from the precomputed BM25 negatives. Each listed negative forms a separate triplet. During asynchronous training, all training queries are encoded before they are divided into five fixed subsets. One subset is searched in each round, so five consecutive rounds cover the complete query set. For each searched query, 20 negatives are sampled from the filtered top-200 results and converted into 20 independent triplets.

3.2.6. GRASS

GRASS (Gap-based Retriever with Adaptive Sample Selection) is an uncertainty-based hard negative selection method with an adaptive candidate pool. Rather than ranking candidates only by similarity, GRASS also considers the model's uncertainty when identifying candidate documents that may provide informative training signals. It avoids the periodic full corpus re-encoding and index rebuilding that ANCE performs by restricting mining to an evolving cached candidate document pool. A utility score records how often each cached document is selected and helps determine which entries are retained, refreshed, or replaced. GRASS also chooses which training queries to revisit. During training, it records for each query the gap between the scores of its mined negative and its positive, and later mining rounds revisit queries whose record gap is large (Section 3.2.6).

The cache architecture is inspired by the Negative Cache streaming method [14] and is adapted for hard-negative uncertainty selection.

Method Overview

GRASS performs negative selection and training as follows:

1. A bounded cache H is initialized by uniformly sampling 32,000 documents from the mining corpus. Stochastic embeddings of these documents are stored for subsequent negative selection.
2. The mining worker selects the queries for the current round. In later rounds, half the places are filled uniformly, and half are reserved for revisiting previously trained queries, with uniform

sampling filling any shortages. Queries with a large recorded gap between the scores of their mined negatives and positives are favored (Section 3.2.6).

3. The mining worker encodes the current training queries and scores each query against documents in H using a combination of mean similarity and uncertainty.
4. Known positive documents are removed, and the highest-scoring remaining document is selected as the query's explicit hard negative.
5. The selected negatives are passed to the training worker. The trainer freshly encodes each query, positive passage, and selected negative before applying the shared contrastive loss.
6. While computing the loss, the trainer records the gap between the scores of each query's mined negative and its positive. These gaps are saved with each checkpoint and used by the miner to select later queries.
7. The cache is maintained periodically. Useful documents with stale embeddings are refreshed, while persistently unused or low utility documents are replaced by stronger candidates from outside the cache.

Figure 3.3 summarizes this pipeline

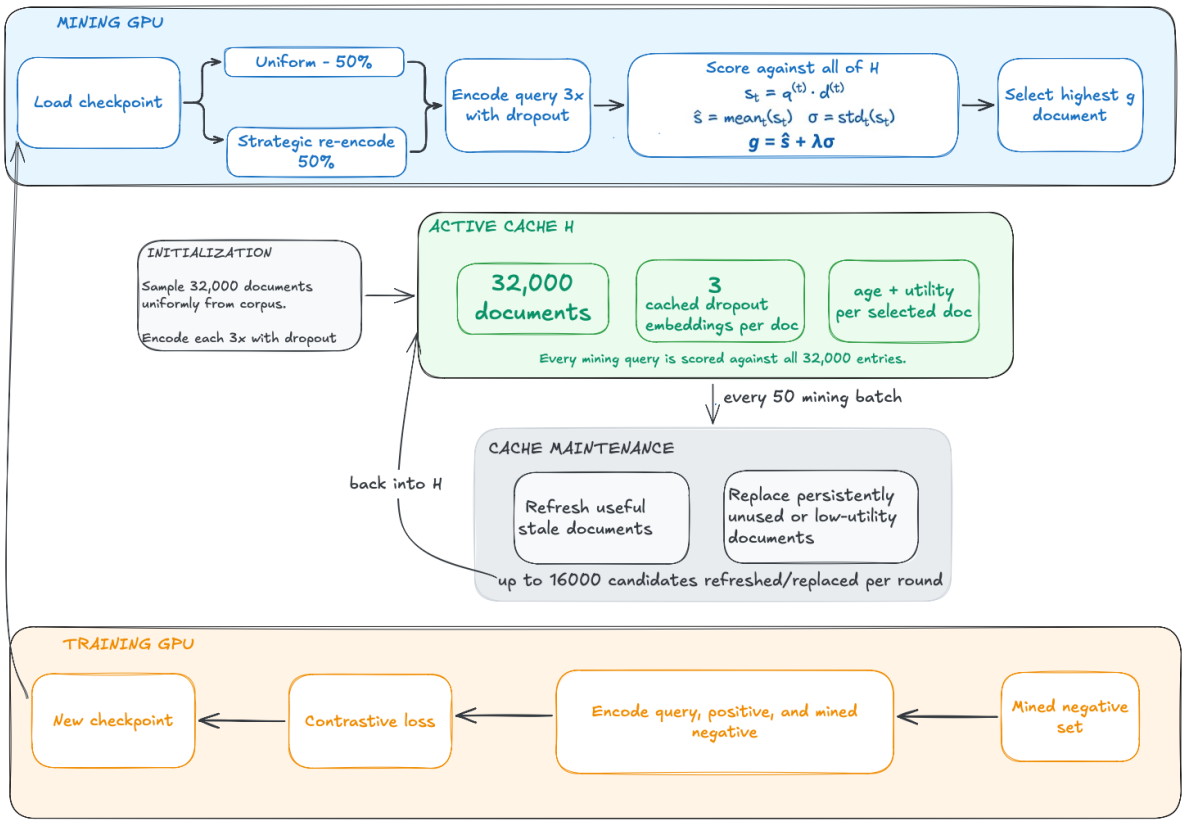

Figure 3.3: GRASS architecture. The mining worker selects uncertainty-guided hard negatives from an evolving document cache, while the training worker updates the retriever using fresh encodings.

Uncertainty-Guided Candidate Score

GRASS estimates uncertainty using Monte Carlo dropout. This approach produces multiple representations of the same input using a single encoder. The implementation uses $T = 3$ encoding passes with dropout enabled at a probability of 0.3. The encoder parameters remain fixed, and no gradients are computed during mining.

Before training begins, the pretrained BGE-M3 model is used to initialize the candidate cache H and produce the first set of hard negatives. A total of 32,000 documents are sampled uniformly from the corpus. Each document is encoded three times, producing three stochastic representations of each of

its segments that are stored in the cache. The training queries are then scored against the initial cache, and training begins with the resulting cache-mined negatives.

During each new mining round, the miner loads the most recent available trainer checkpoint. Unlike ANCE, it does not re-encode the complete corpus or rebuild a FAISS index. Instead, each query in the current batch is encoded three times and compared with all 32,000 cached documents using inner-product similarity. The stored document representations are recomputed only when their documents are selected for refresh or replacement during cache maintenance.

For each query q and cached document d, the similarity during stochastic pass t is calculated as follows:

$$s_t(q,d) = \max_k \mathbf{q}^{(t)} \cdot \mathbf{d}_k^{(t)}, \qquad t \in \{1,\dots,T\}. \tag{3.3}$$

Where $\mathbf{q}^{(t)}$ is a query representation computed during the current mining round, and $\mathbf{d}_k^{(t)}$ is the representation of segment k of document d from pass t, already stored in H. This is the score of Equation 3.1 computed within one dropout pass. The query and document are encoded independently. For each value of t, one newly computed query representation is compared with the stored segment representations from the same pass, producing three similarity samples for each query–document pair.

The mean similarity is calculated as

$$\hat{s}(q,d) = \frac{1}{T}\sum_{t=1}^{T} s_t(q,d). \tag{3.4}$$

and the variation across the three samples is measured using the standard deviation:

$$\sigma(q,d) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(s_t(q,d) - \hat{s}(q,d)\right)^2}. \tag{3.5}$$

A larger value of $\sigma(q,d)$ indicates that the similarity assigned to the query–document pair is less stable under different dropout masks. GRASS therefore uses $\sigma(q,d)$ as an estimate of model uncertainty during negative selection.

The final candidate score is calculated as follows:

$$g(q,d) = \hat{s}(q,d) + \lambda\sigma(q,d). \tag{3.6}$$

Where λ controls the contribution of uncertainty. When $\lambda = 0$, candidates are ranked only by their mean similarity. Increasing λ gives greater preference to documents whose similarity to the query varies more across the dropout passes. After known positive documents are removed, the remaining cached documents with the highest $g(q,d)$ are selected as the query's explicit hard negative.

Query Selection

GRASS uses 5,158 optimizer updates at a batch size of 64, corresponding to approximately half the query budget of two epochs over the full training mixture. The initial negatives are mined for a fixed sample of 164,996 queries. These are sampled from MS MARCO, ReasonIR VL, and ReasonIR HQ in proportion to their sizes. Within each source, the queries are sorted by length and split into ten equally sized length bins, and the same fraction is sampled from each bin. The sample therefore has the same source and length composition as the full training set. Later mining rounds select their queries from all 329,993 training queries, half uniformly and half based on gaps recorded during training, as described below.

To decide which queries to revisit, the trainer records a gap for every query it trains on. The gap is the signed difference between the similarity of the query to its mined negative and to its positive document,

$$\Delta(q) = s(q,p_q^-) - s(q,p_q^+), \tag{3.7}$$

where both similarities are the MaxP scores (Equation 3.1) computed during the contrastive loss, before division by the temperature. A larger positive gap indicates a query that the model currently finds harder. Only the most recent gap of each query is kept and saved with each checkpoint, so a gap remains unchanged until its query is trained again.

Each later mining round selects approximately 16,500 queries. The number taken from each source and length bin is proportional to its size in the full training set, so each round keeps the composition of the training data. Within each bin, half of the queries are sampled uniformly. The other half are selected through replacement, which revisits queries that already have a recorded gap. These candidates are ranked from the largest to the smallest gap, and each receives the weight.

$$w(q) = \mathrm{rank}(q)^{-\alpha}, \qquad \alpha = 0.7. \tag{3.8}$$

Queries with larger gaps are therefore more likely to be replaced, while every candidate keeps some chance of selection. The selected queries are then mined as described above, so a replayed query receives a fresh negative from the current model and cache.

The selection rule follows loss-based example selection. As in Online Batch Selection, each query is ranked by its most recent gap [16]. The rank-based weights, including $\alpha = 0.7$, follow Prioritized Experience Replay [23], without its importance-sampling correction. A persistently large gap may also reflect a relevant but unlabeled negative or an unreliable training example. A query can therefore be selected through replay at most twice.

Cache Maintenance

The cache is updated because its document representation becomes outdated as the model changes, and its candidate set must adapt during training. While generating a negative set, the miner performs one cache-maintenance event after every 3200 processed queries. It updates the cache and then continues mining the remaining queries against the modified cache. This schedule is used both for the initial negative set and for subsequent mining rounds.

For every cached document d, GRASS records whether it has been selected at least once since the previous maintenance event. Its utility is updated using:

$$u_d \leftarrow 0.95u_d + 0.05a_d, \tag{3.9}$$

where $a_d = 1$ if the document was selected during the latest interval and $a_d = 0$ otherwise. A document becomes eligible for replacement if it has not been selected during three consecutive maintenance intervals, or if the utility is at most 0.01 and its representations are at least 150 trainer updates old.

Each maintenance event refreshes useful documents whose representations are outdated and replaces low-utility documents with new documents from the corpus. A refresh recomputes the three stochastic representations of a document with the current model. A document is outdated when more than about 500 trainer updates have passed since its representations were last computed. Later in training, the model changes more slowly, and the maintenance budget is smaller, so this limit is gradually relaxed to 1,000 updates. Outdated documents are handled first. Any remaining capacity replaces other low-utility documents and then refreshes the remaining useful documents. These are ranked by their utility multiplied by the age of their representations, so a document is refreshed earlier the more often it is selected, and the longer ago its representations were computed.

Refreshes and replacements share a budget that decreases linearly from 16,000 to 8,000 per mining round. The largest early budget keeps the cache up to date while the model changes quickly. The age threshold equals the number of training updates that the budget needs to cover the whole cache once, so it grows as the budget decreases. Each cache position is refreshed or replaced at most once per mining round, so a newly inserted document has a chance to be selected before it can be replaced again.

For n cache positions selected for replacement, $2n$ candidate documents are sampled uniformly from the corpus outside of H. Each candidate is encoded three times and scored separately against up to

128 recent queries using $g(q, d)$ with $\lambda = 0$, producing one score per query. Known positive query–document pairs are excluded. The five highest query-specific scores for each candidate are averaged to obtain its admission score. The n candidates with the highest admission scores are inserted into the selected cache positions.

Maintenance takes place while a mining round is being processed. Later queries in the same round may therefore be scored against a cache that has already been partially refreshed or updated.

Asynchronous Mining and Training Schedule

GRASS runs training and mining concurrently on two A100 GPUs. It follows the asynchronous process described for ANCE in Section 3.2.5 but replaces full-corpus encoding and FAISS index construction with negative selection from the bounded cache H.

The training worker runs on GPU 0 and owns the model parameters, optimizer, and learning-rate scheduler. It saves a checkpoint every 250 optimizer updates. The mining worker runs on GPU 1 and owns the document cache and its maintenance state. At the start of each mining round, it loads the newest saved checkpoint, skipping any intermediate ones, and keeps it fixed while it encodes the round's queries, selects their negatives, and maintains the cache. The initial negative set, mined with the pretrained model before training, is not counted as a mining round.

After a round, the miner publishes one training record per selected query, pairing the query and its positive passage with the newly selected negative. The trainer checks for new records every 50 optimizer updates and keeps training on its current records until they arrive. It then replaces the active records with the new ones. Training batches are re-formed with the source-based batching of Section 4.3.1 whenever the active records are exhausted or replaced.

4

Experimental Setup

This chapter lists the experiments reported in the next chapter, the computational environment, and the training settings of each experiment. The methods and data are described in the Methodology chapter.

Two experimental settings must be distinguished. The main comparison evaluates BGE-M3 retrievers on BRIGHT using a shared training mixture and encoding configuration. A separate experiment evaluates a RoBERTa-based, paper-faithful ANCE implementation on MS MARCO Passage. The MS MARCO experiment therefore uses a different model configuration and training dataset from the BRIGHT experiments.

4.1. Experiments

Table 4.1 lists the evaluated models. Each row states what distinguishes the model and refers to the section that describes it.

Table 4.1: Evaluated models, grouped by experimental setting. In the BRIGHT setting, the trained models fine-tune BGE-M3 on the training mixture (Table 3.4), and all models are evaluated on the 12 BRIGHT domains. In the MS MARCO setting, RoBERTa-base is trained on MS MARCO Passage and evaluated on MS MARCO Dev.

Model	What distinguishes it	Described in
BRIGHT setting		
BM25	Sparse lexical retrieval, no training	Table 3.7
Base BGE-M3	Pretrained model, no fine-tuning	Table 3.1
In-batch	Predefined hard negative and in-batch negatives	Section 3.2.3
Cross-batch	In-batch negatives pooled across GPUs	Section 3.2.4
ANCE	Hard negatives mined from the full corpus	Section 3.2.5
GRASS $\lambda = 0, 0.3, 0.5$	Hard negatives mined from an adaptive cache	Section 3.2.6
MS MARCO setting		
ANCE validation	Original ANCE objective and optimizer	Section 3.2.5

Three GRASS models are trained with seed 42 from the same initial query half, cache initialization, and training budget. All three use the gap-based query selection described in Section 3.2.6, with λ set to 0, 0.3, and 0.5.

4.2. Computational Environment

4.2.1. Hardware: DelftBlue Cluster

All experiments were conducted on the DelftBlue high-performance computing cluster at TU Delft. Resources were allocated according to the computational requirements of each retrieval method [1].

For sparse retrievers, experiments were run on Phase 1 CPU nodes in the `compute` partition. Each node provides 48 CPU cores and 185 GB of RAM.

For dense retrievers, experiments were executed on Phase 2 nodes in the `gpu-a100` partition. Each node contains four NVIDIA A100 GPUs with 80 GB of VRAM per GPU. Individual jobs used one or two GPUs depending on the training strategy. Multi-GPU nodes were used either for Distributed Data Parallel (DDP) training, which synchronized gradients across workers in the cross-batch experiment, or to execute training and negative mining concurrently on separate GPUs in the ANCE and GRASS pipelines.

4.2.2. Software and Libraries

The implementation uses PyTorch and HuggingFace Transformers for neural-model training and inference. Tevatron reads the training data and loads and saves the dual-encoder models [10]. It is used by the in-batch, cross-batch, and BGE-M3 ANCE pipelines. The remaining components were implemented for this thesis, including data preprocessing, document segmentation and MaxP scoring, hard-negative mining, FAISS retrieval, BRIGHT exclusion filtering, and metric computation.

GradCache was used by all BGE-M3 training strategies to support a large batch under GPU memory constraints. Its mechanism is described in Section 3.2.4.

The BM25 baseline is implemented with Pyserini and Lucene [13]. Pyserini constructs and searches the sparse indexes, while the evaluation pipeline applies the BRIGHT exclusion protocol and computes the final metrics.

Dense retrieval uses exact inner-product search through FAISS [6]. Both sparse and dense rankings are evaluated with `pytrec_eval` [26].

4.3. Training Settings

This section lists the training settings of each experiment.

4.3.1. Shared Training Setup

The BRIGHT training strategies use AdamW with a learning rate of 1×10^{-5}, a warm-up ratio of 0.1, weight decay of 0.01, and gradient clipping at 1.0. The in-batch, cross-batch, and ANCE strategies train for two epochs. GRASS uses a smaller budget, described in Section 3.2.6.

4.3.2. In-Batch

A full batch contains 64 queries, each paired with one positive passage and one predefined hard negative passage, producing 128 passages. Each query is therefore trained against all 128 passages, consisting of one positive passage and 127 negatives. One is the query's predefined hard negative, and the remaining 126 are the positive and negative passages belonging to the other 63 queries. Training runs on one GPU for two epochs, resulting in 10,312 optimizer updates.

4.3.3. Cross-Batch

Training uses two NVIDIA A100 GPUs, with one Distributed Data Parallel worker per GPU. Each worker processes 512 queries, and every query is paired with one positive and one predefined hard negative. Each GPU therefore processes 512 queries and 1,024 passages. Tevatron gathers the embeddings from both workers before computing the distributed contrastive loss [10]. The resulting loss contains 1,024 queries and 2,048 passages.

Each query is compared with its positive passage and 2,047 negatives. These negatives consist of its own predefined hard negatives, the 1,023 positive passages belonging to the other queries in the combined batch, and the 1,023 predefined hard negatives paired with those queries.

GradCache processes queries and passages in chunks of 32, while trainer-level accumulation is set to one. Gradient checkpointing further reduces memory use by recomputing intermediate encoder activations during backpropagation instead of retaining them throughout the forward pass. All chunks from both GPUs contribute to one contrastive loss. Their gradients are accumulated and synchronized across two GPUs before one optimizer update is performed.

With 329,993 training records and 1,024 queries per update, each epoch contains

$$\left\lfloor \frac{329{,}993}{1{,}024} \right\rfloor = 322$$

optimizer updates. Training for two epochs therefore produces 644 optimizer updates. The incomplete final batch of each epoch is dropped, so every update uses the same negative-pool size.

Table 4.2 summarizes the batch composition, negative-pool size, and number of optimizer updates for the in-batch and cross-batch baselines.

Table 4.2: Full-batch composition of the in-batch and cross-batch baselines.

Method	Queries per update	Passages per update	Predefined negatives per query	Total negatives per query	Updates over two epochs
In-batch	64	128	1	127	10,312
Cross-batch	1,024	2,048	1	2,047	644

Both baselines process the same training dataset for two epochs. However, the cross-batch configuration uses both a larger query batch and a larger negative pool. Processing 1,024 rather than 64 queries per update also reduces the number of optimizer updates from 10,312 to 644.

4.3.4. ANCE for BRIGHT

Training uses a batch size of 64 and a group size of two. As in the in-batch baseline, each query is compared with one positive passage and 127 negatives. Training lasts for two epochs, corresponding to 10,312 optimizer updates. The completed run used nine refreshed negative sets mined from distinct checkpoints.

4.3.5. GRASS

A GRASS run is used only if all 5,158 updates complete, the final model is saved, and the mining worker does not fail. At least three consumed rounds must come from distinct checkpoints after initialization and include cache refreshes. Each qualifying round must be trained on for at least 128 updates, and together these rounds must cover at least half of the training updates.

A GRASS run is reported only if its model was mainly trained on negatives from the adaptive mining loop, rather than on the initial negatives. The run must complete all 5,158 updates and save its final model, and the miner must not fail. At least three mining rounds, each mined with a different checkpoint, must include cache refreshes and be trained on for at least 128 updates each. Together, these rounds must be trained on for at least half of all updates.

4.3.6. MS MARCO ANCE Validation

The warm-up uses a batch size of 32, a learning rate of 2×10^{-4}, and a linear learning-rate scheduler with 1,000 warm-up updates. ANCE training uses a batch size of 64, a learning rate of 1×10^{-6}, and a linear learning-rate scheduler with 5,000 warm-up updates. The training GPU saves a checkpoint every 10,000 updates, and the mining GPU uses the newest saved checkpoint after completing its current round. Training is limited to 300,000 updates rather than the reported 600,000-update checkpoint due to hardware constraints. The original setup used four training GPUs and four mining GPUs, whereas this experiment uses one GPU for each task. The experiment is therefore treated as an implementation validation rather than an exact reproduction.

4.4. Comparisons

All BRIGHT models are evaluated on the same 12 domains. Recall@1000 is the primary metric, and NDCG@10 and MRR are secondary metrics. Differences between models are tested with the paired t-test described in Section 3.1.

The comparisons of GRASS $\lambda = 0.3$ and GRASS $\lambda = 0.5$ against GRASS $\lambda = 0$ measure whether uncertainty helps. Because these two comparisons are tested together, each uses a Bonferroni-corrected

Table 4.3: GRASS-specific settings used in the three experimental models.

Property	Configuration
Candidate-cache size	32,000 documents
Monte Carlo dropout passes	3
Dropout probability	0.3
Uncertainty weights	0, 0.3, 0.5
Cache-admission uncertainty weight	0
Selection rule	Highest $g(q, d)$
Explicit negatives per query	1
Initial query set	164,996 queries
Queries per later round	Approximately 16,500
Replay fraction	0.5
Replay weighting	$\mathrm{rank}^{-0.7}$ of the gap
Maximum replay admissions per query	2
Maintenance interval	3,200 mined queries
Age threshold for prioritizing refresh	$500 \rightarrow 1{,}000$ trainer updates
Maintenance budget per event	$3{,}200 \rightarrow 1{,}600$ cache positions
Recent-query reservoir	128 queries
Checkpoint interval	250 optimizer updates
Completed-record check	Every 50 optimizer updates
Consumed mining rounds	19
Training duration	5,158 optimizer updates

significance threshold of $p < 0.05/2$ instead of 0.05. Selected negatives shape later gaps and therefore later query choices, so these comparisons measure the whole adaptive loop rather than negative selection alone.

For the GRASS models, the selected negatives are also compared across the three values of λ. Training cost is compared by elapsed time and allocated GPU-hours. The MS MARCO model is compared with the published ANCE result, as described in Section 3.1.7.

5

Results

This chapter presents retrieval results on the 12 BRIGHT domains for BM25; base BGE-M3 before fine-tuning; in-batch and cross-batch training; ANCE-style mining; and three GRASS settings. Recall@1000 is the main metric. NDCG@10 and mean reciprocal rank (MRR) are secondary metrics. Scores are reported for each domain and as averages over the domains. The later sections compare the training strategies, the effect of uncertainty on GRASS negative selection, and the training and mining time. The last section checks the local ANCE implementation against the published MS MARCO result. The Discussion chapter interprets the results.

5.1. Overall Retrieval Performance

Table 5.1 shows the average scores across the 12 BRIGHT domains. ANCE achieves the highest Recall@1000 (0.6737) and NDCG@10 (0.1373). Base BGE-M3 has the highest MRR (0.1946), while GRASS $\lambda = 0$ has the highest MRR among the trained models. For Recall@1000, none of the trained models perform significantly better than base BGE-M3 (0.6650). Although ANCE scores 0.0088 higher, this difference is not statistically significant ($p = 0.26$). In-batch, on the other hand, performs significantly worse than base BGE-M3 ($p < 0.001$).

The three GRASS settings achieve Recall@1000 scores ranging from 0.6591 to 0.6692, with $\lambda = 0.5$ obtaining the highest score. None differs significantly from base BGE-M3 ($p > 0.4$). GRASS with $\lambda = 0$ however, performs significantly better than in-batch ($p < 0.001$). Cross-batch obtains a Recall@1000 score of 0.6526, lower than base BGE-M3 but higher than BM25. In-batch has the lowest scores across all three metrics.

Table 5.1: Retrieval performance averaged across the 12 BRIGHT domains. Base BGE-M3 denotes the model before fine-tuning. Bold indicates the highest score for each metric.

Method	Optimizer updates	Recall@1000	NDCG@10	MRR
BM25	–	0.6397	0.1369	0.1892
Base BGE-M3	–	0.6650	0.1344	**0.1946**
In-batch	10,312	0.6171	0.1219	0.1642
Cross-batch	644	0.6526	0.1319	0.1792
ANCE-style mining	10,312	**0.6737**	**0.1373**	0.1843
GRASS $\lambda = 0$	5,158	0.6681	0.1327	0.1866
GRASS $\lambda = 0.3$	5,158	0.6591	0.1341	0.1803
GRASS $\lambda = 0.5$	5,158	0.6692	0.1336	0.1844

5.2. Performance Across BRIGHT Domains

Table 5.2 reports Recall@1000 for each BRIGHT domain. A trained model achieves the highest score across all methods in six domains. GRASS $\lambda = 0.3$ achieves the highest score in Biology, while GRASS

$\lambda = 0$ performs best in Psychology. GRASS $\lambda = 0.5$ leads in Stack Overflow and Sustainable Living. ANCE has the highest score in TheoremQA–Theorems and in-batch in TheoremQA–Questions. BM25 performs best in Earth Science, Pony, LeetCode, and AoPS, while base BGE-M3 leads in Economics and Robotics. All trained models outperform base BGE-M3 in LeetCode and both TheoremQA domains. ANCE and the three GRASS settings also exceed it in Biology, and the three GRASS settings in Psychology. ANCE and GRASS $\lambda = 0.5$ exceed it in Stack Overflow, cross-batch and GRASS $\lambda = 0.3$ and $\lambda = 0.5$ in Sustainable Living, and GRASS $\lambda = 0.3$ in Earth Science.

Pony is the domain with the largest drop for every trained model. Base BGE-M3 scores 0.7636, whereas the trained models range from 0.3822 to 0.6025. GRASS $\lambda = 0$ has the highest score among the trained models. Excluding Pony, ANCE has the highest average score (0.6929). GRASS $\lambda = 0.5$ follows at 0.6875, and GRASS $\lambda = 0.3$ at 0.6823. The scores for GRASS $\lambda = 0$ cross-batch, base BGE-M3, in-batch, and BM25 are 0.6740, 0.6679, 0.6560, 0.6385, and 0.6251, respectively. The results averaged over all 12 domains are still used as the main results. Table 5.3 shows Pony Recall@1000 separately for tutorial pages and reference pages, where all trained models score below base BGE-M3 on tutorial pages and above it on reference pages.

Table 5.2: Recall@1000 across the 12 BRIGHT domains. BGE-M3 refers to the model before fine-tuning. Bold indicates the highest score among the trained dense retrievers for each domain.

Domain	BM25	BGE-M3	In-batch	Cross-batch	ANCE	GRASS $\lambda = 0$	GRASS $\lambda = 0.3$	GRASS $\lambda = 0.5$
Biology	0.7644	0.7566	0.5955	0.7169	0.8233	0.8267	**0.8432**	0.8354
Earth Science	0.8101	0.7769	0.6946	0.7748	0.7642	0.7685	**0.7772**	0.7733
Economics	0.6833	0.7677	0.6501	0.7277	0.7119	0.7122	**0.7515**	0.7380
Psychology	0.6212	0.8094	0.7044	0.7699	0.7822	**0.8183**	0.8133	0.8160
Robotics	0.6582	0.6880	0.6107	0.6279	0.6687	0.6471	0.6441	**0.6798**
Stack Overflow	0.6735	0.7355	0.7206	0.7308	0.7480	0.7217	0.7278	**0.7562**
Sustainable Living	0.7424	0.7719	0.7320	0.7766	0.7592	0.7714	0.7856	**0.7898**
Pony	0.8007	0.7636	0.3822	0.4834	0.4631	**0.6025**	0.4039	0.4676
LeetCode	0.8177	0.6614	0.7853	0.7596	**0.7988**	0.7883	0.7789	0.7678
AoPS	0.3971	0.3754	0.3381	0.3548	**0.3652**	0.3572	0.3636	0.3539
TheoremQA–Theorems	0.3846	0.4262	0.6183	0.5777	**0.6446**	0.4903	0.4823	0.5305
TheoremQA–Questions	0.3230	0.4471	**0.5733**	0.5306	0.5557	0.5128	0.5382	0.5217

Table 5.3: Pony Recall@1000 by type of relevant page. Tutorial pages explain basic Pony syntax. Reference pages document specific library features.

Page type	BGE-M3	In-batch	Cross-batch	ANCE	GRASS $\lambda = 0$	GRASS $\lambda = 0.3$	GRASS $\lambda = 0.5$
Tutorial pages	0.77	0.37	0.47	0.45	0.60	0.39	0.46
Reference pages	0.67	0.74	0.74	0.73	0.77	0.75	0.72

5.3. Paired Comparisons Between Training Strategies

Table 5.4 shows the mean Recall@1000 difference and the wins, losses, and ties between the two methods over the 12 domains. ANCE overall scores 0.0566 higher than in-batch, with 11 wins and 1 loss. Cross-batch scores 0.0355 higher than in-batch, with 9 wins and 3 losses. Both differences are statistically significant. GRASS $\lambda = 0$ also performs better than in-batch by 0.0510 with 10 wins, but performs worse than ANCE by 0.0057 with six wins. It performs better than cross-batch by 0.0155, also with six wins. Of these three, only the difference with ANCE is not statistically significant.

5.4. Effect of Uncertainty-Guided Selection

5.4.1. Retrieval Performance Across Uncertainty Weights

Table 5.5 compares the three GRASS settings. Average Recall@1000 is 0.6681 at $\lambda = 0$, 0.6591 at $\lambda = 0.3$, and 0.6692 at $\lambda = 0.5$. $\lambda = 0.3$ and $\lambda = 0.5$ perform better than $\lambda = 0$ in seven and eight of the 12 domains, respectively.

5.4.2. Effect on Negative Selection

Table 5.6 shows the results from the 19 mining rounds after the initial negative preparation. Each setting selects 313,493 query-level negatives across these rounds, including repeated queries.

Table 5.4: Average paired differences in Recall@1000 across the 12 BRIGHT domains. Positive values favor the first method. W/L/T indicates the number of domain wins, losses, and ties. P-values are based on a two-sided paired t-test, with equal weight given to each domain. The two comparisons between GRASS settings use a Bonferroni-corrected significance threshold of 0.05/2

Comparison	Mean difference	p-value	W/L/T
Cross-batch − in-batch	$+0.0355$	$p < 0.001$	9/3/0
ANCE − in-batch	$+0.0566$	$p < 0.001$	11/1/0
GRASS $\lambda = 0$ − in-batch	$+0.0510$	$p < 0.001$	10/2/0
ANCE − GRASS $\lambda = 0$	$+0.0057$	$p = 0.345$	6/6/0
GRASS $\lambda = 0$ − cross-batch	$+0.0155$	$p = 0.022$	6/6/0
GRASS $\lambda = 0.3 - \lambda = 0$	-0.0089	$p = 0.073$	7/5/0
GRASS $\lambda = 0.5 - \lambda = 0$	$+0.0011$	$p = 0.843$	8/4/0

Table 5.5: Average retrieval scores across the 12 BRIGHT domains for the three GRASS settings. Recall@1000 differences and domain wins, losses, and ties are relative to $\lambda = 0$. Differences are calculated before the average scores are rounded for display.

λ	Recall@1000	NDCG@10	MRR	Δ Recall@1000	Wins/losses/ties
0	0.6681	0.1327	0.1866	–	–
0.3	0.6591	0.1341	0.1803	-0.0089	7/5/0
0.5	0.6692	0.1336	0.1844	$+0.0011$	8/4/0

When uncertainty is added to the score, the selected negative changes for 13.5% of queries at $\lambda = 0.3$ and 22.9% at $\lambda = 0.5$. The average uncertainty σ of the selected negatives is 0.0244 at $\lambda = 0.3$ and 0.0282 at $\lambda = 0.5$, against 0.0221 at $\lambda = 0$. The age of the selected negatives and the numbers of cache refreshes and replacements are similar in all three settings (Table 5.6).

Table 5.6: GRASS negatives at each λ over the 19 mining rounds, averaged over mined queries. A flip is a query whose negative changes when $\lambda\sigma$ is added.

λ	Flip rate	Selected $\hat{s}$	Selected σ	Selected $\lambda\sigma$	Age (steps)	Refreshes	Replacements
0	0.0%	0.5388	0.0221	0	246	15,168	261,228
0.3	13.5%	0.5357	0.0244	0.0073	246	15,911	260,485
0.5	22.9%	0.5451	0.0282	0.0141	237	14,720	261,676

5.5. Training and Mining Efficiency

Before training, ANCE and GRASS mine an initial set of negatives with base BGE-M3 (Table 5.7). For GRASS, this step takes 47 minutes, 15 minutes longer than for ANCE, whose 32 minutes also include job startup.

GRASS $\lambda = 0$ finishes training in 6 hours 29 minutes, almost four hours before ANCE. Cross-batch is the only baseline that finishes sooner. GRASS trains on half as many query examples as ANCE and on 42.3% of the distinct training queries.

Table 5.8 compares the training budgets and the work performed during each negative refresh. ANCE mines negatives for all 329,993 training queries against the full 648,942-document corpus. GRASS mines approximately 16,500 queries per round, half sampled uniformly and half replayed, against its 32,000-document cache. ANCE rounds average 35.2 minutes. GRASS rounds average 11.3 minutes. ANCE spends 19.5 minutes of each round encoding the full corpus, and GRASS $\lambda = 0$ spends 2.4 minutes of each round on cache maintenance. Each round covers a different amount of work. Mining overlaps with training, so refresh times are included in the total elapsed times in Table 5.7.

As Table 5.8 shows, GRASS $\lambda = 0$ used almost all of its mined negatives in training (95.1% vs. 18.8% for ANCE), and these negatives were more recent and took less time to mine (3.6 vs. 5.3 hours).

Table 5.7: Time and GPU cost of each run, rounded to the nearest minute. Preparation does not include dataset preprocessing or queue time.

Method	GPUs	Updates	Query examples	Preparation	Training	Total	GPU-hours	Recall@1000
In-batch	1	10,312	659,968	–	11:39	11:39	11.6	0.6171
Cross-batch	2	644	659,456	–	4:26	4:26	8.9	0.6526
ANCE-style mining	2	10,312	659,968	0:32	9:52	10:24	20.8	0.6737
GRASS $\lambda = 0$	2	5,158	330,112	0:47	5:42	6:29	13.0	0.6681
GRASS $\lambda = 0.3$	2	5,158	330,112	0:47	5:46	6:33	13.1	0.6591
GRASS $\lambda = 0.5$	2	5,158	330,112	0:47	5:15	6:01	12.0	0.6692

Table 5.8: Training time and mining work of ANCE and GRASS, with GRASS timings averaged over its three runs and the last four rows from GRASS $\lambda = 0$.

	BGE-M3 ANCE	GRASS
Training budget	2 epochs, 10,312 steps	5,158 steps
Checkpoint cadence	Every 1,000 steps	Every 250 steps
Queries mined per refresh	All 329,993	Approximately 16,500
Documents searched	Full corpus, 648,942	32,000-document cache
Fresh rounds consumed	9	19 per run
Mining per refresh round, mean	35.2 min	11.3 min
Elapsed time per 1,000 training steps, mean	57.2 min	64.2 min
Mined negatives used in training	557,568 of 2,969,937 (18.8%)	298,112 of 313,493 (95.1%)
Checkpoint age of negatives when used	600–700 updates	200–250 updates
Document-side encodings over refreshes	6,427,314	1,777,698
Mining time over refreshes, total	5.3 h	3.6 h

5.6. MS MARCO ANCE Reference Comparison

On the MS MARCO passage development set, the local ANCE model achieved an MRR@10 of 0.2699 and a Recall@1000 of 0.9029. Microsoft's published ANCE FirstP scores[1] are 0.3300 and 0.9590, respectively. Table 5.9 reports these results alongside the optimizer update of each evaluated checkpoint.

Table 5.9: ANCE results on the MS MARCO passage development set. The local evaluation contains 6,980 judged queries.

Model	Checkpoint update	MRR@10	Recall@1000
Microsoft ANCE FirstP, published	600,000	0.3300	0.9590
Local ANCE	300,000	0.2699	0.9029

[1] Microsoft ANCE repository: `https://github.com/microsoft/ANCE`.

6

Discussion

6.1. Main Findings

RQ1 aimed to check whether the proposed architecture is sample efficient in finding hard negatives without relying on large batch sizes and can perform similarly well to other existing methods. GRASS scored 0.6681 on Recall@1000 across 12 BRIGHT domains, compared with 0.6737 for ANCE and outperformed in-batch and cross-batch models. GRASS used only 64 queries per update, while cross-batch used 1,024. However, none scored significantly above the baseline. Thus, this dataset did not generally benefit from fine-tuning. GRASS was also more efficient in utilizing its negatives, using 95.1% compared with ANCE, which used only 18.8% before its next refresh. Adding uncertainty, however, did not significantly change the retrieval score.

RQ2 aimed to check whether the proposed architecture can reduce the training time and cost of dense retrievers while still performing similarly well to other existing methods. GRASS finished training almost four hours before ANCE, although it also trained on half as many query examples. Each GRASS mining round took about a third of the time of an ANCE round, and GRASS still scored similarly to ANCE.

Without Pony, all models except in-batch scored above base BGE-M3. GRASS was the most stable across domains, with the smallest score deviations from the baseline, most evident on Pony (Table 5.2).

6.2. Sample-Efficient Training Without Large Batches or Global Mining

6.2.1. Cache-Based Mining Without Large Batches

The results suggest that hard negatives can provide a strong training signal without large batches. Cross-batch training processes 1,024 queries per update to enlarge its negative pool. GRASS processes 64 queries and draws negatives from its cache. Despite the smaller batch, GRASS scores above cross-batch and on par with ANCE (Section 6.1). This suggests that the difficulty of the negatives matters more than the size of the negative pool.

GRASS uses almost all the negatives it mines per query before the next refresh. ANCE, however, re-encodes the whole corpus every mining round but only manages to use 18.8% of its mined negatives before the next refresh replaces them. Because each GRASS round mines only a subset of the queries, it is cheap enough to run four times as often as an ANCE refresh, each time with a more recent checkpoint. As a result, when GRASS trains on a negative, the checkpoint that mined it is only 200 to 250 updates older than the model being trained, compared with 600 to 700 updates for ANCE (Table 5.8). The ANCE authors call this delay between mining and training the asynchronous gap. They found that refreshing the index too rarely made the training loss and retrieval accuracy fluctuate instead of converging, as the outdated negatives can drive the model towards undesired local optima [28]. GRASS's negatives therefore reflect what the model still confuses later in training.

GRASS also trained on half as many query examples as ANCE (Table 5.7) and still performed on par

with two-epoch ANCE on BRIGHT. The model therefore did not need to see every training query to match two-epoch ANCE. However, a one-epoch ANCE run was not performed for comparison (Section 6.5).

6.2.2. Uncertainty-Guided Negative Selection

GRASS hard negative selection was also tested with an uncertainty term, expecting that combining it with similarity would find the hardest document for each query. Recall did not improve, even though a sizable share of the selected negatives changed when λ went from 0.5 to 0.3, and the variance score grew as λ increased (Table 5.6). So the uncertainty signal did not help find harder negatives.

One possible reason the uncertainty did not help is that it could only swap a negative for a slightly less similar one. This is because the negative documents had similarity scores of about 0.54, while their σ was only about 0.02 to 0.03 (Table 5.6), so a document could only overtake the most similar one if their similarity scores were almost equal and its σ made up the small difference. In the contrastive loss, a lower similarity score means a smaller weight, so the swapped negative had less effect on training than the original one. As a result, when uncertainty changed the pick, it removed the query's hardest negative, which is the one the model learns the most from, and replaced it with a slightly easier one. This happened for only a minority of queries, so training was almost the same as without uncertainty, and retrieval did not improve significantly (Table 5.4). The domains separately also did not provide any clear benefits from incorporating uncertainty (Table 5.2). This result concerns uncertainty only, as cache-based mining with similarity alone performed on par with ANCE (Section 6.2.1). Future work should consider incorporating uncertainty into the loss itself, so that an uncertain negative receives more weight during training.

Another issue could be using MC Dropout for uncertainty estimation. GRASS used three dropout passes, as increasing it too much would increase the mining duration. MC Dropout also gave very small σ values.

A better alternative, which could give uncertainty values large enough to produce more drastic changes rather than the order between the nearest candidates based on similarity score, is an ensemble of encoders that averages the similarity score across encoders and uses the spread between their scores as the uncertainty. The encoders can be separately trained models or, to avoid training several methods, earlier checkpoints from the same training run.

6.3. Training Cost and Retrieval Performance

6.3.1. Measured Cost at Competitive Retrieval

When looking only at the time of a mining round, GRASS was much cheaper than ANCE, as presented in Table 5.8. This is because it re-encoded only part of the documents in its cache each round, rather than re-encoding the whole corpus per round. This greatly reduced the time cost, as re-encoding the corpus took more than half of each ANCE round, while cache maintenance took only a few minutes per GRASS round. It was not only less costly but also much more resource-efficient. ANCE spent significant time re-encoding the whole corpus every round, only to do it again by the next refresh, wasting most of the negatives it had mined. GRASS, however, used almost all of its mined negatives before the next mining round, since each round mined about as many queries as the trainer used before the next refresh.

GRASS was also able to find good quality hard negative documents despite its small cache size. This shows that the cache converged to a subset of the most relevant documents for the query sets. The cache started as a uniform sample of the corpus, but its contents changed several times during training, as shown in Table 5.6. In addition, the documents it selected were, on average, encoded at only one checkpoint, or about 240 updates, before the checkpoint used for mining. This means that GRASS kept the embeddings of the documents it used close to the current model, without having to re-encode the ones it did not use.

With two GPUs, however, GRASS's faster mining did not actually reduce total training time. Mining ran on a separate GPU, and because of the small dataset size, the mining GPU was not an actual bottleneck. Thus, in this case, more frequent mining rounds were performed for GRASS to make use

of the mining GPU. GRASS's shorter total training time, as shown in Table 5.7, mainly came from training for half as many updates as ANCE. Faster mining let GRASS mine new negatives more often and train on more recent ones. On a single GPU, mining and training would take turns, and the faster mining would then also shorten the total training time.

6.3.2. Expected Training Time on a Single GPU and Larger Corpora

On a single GPU, mining cannot overlap with training and adds directly to the total time. In this case, GRASS would benefit from its cheaper rounds, spending less total time mining than ANCE over the measured schedule despite refreshing about twice as often (Table 5.8).

The difference in mining time would likely grow on a larger corpus. ANCE encodes the whole corpus in every mining round, which on BRIGHT already took 19.5 of the 35.2 minutes per round. A GRASS mining round depends only on the cache size, the number of mined queries, and the maintenance budget, none of which grow with the corpus. If an ANCE mining round took longer than the 1,000 updates between checkpoints, the trainer would train on older negatives, which can make training fluctuate.

6.4. Robustness Across Domains

Fine-tuning helped most in the domains where the evaluation task resembles the training pairs. In the HQ part of the training data, the positive is a BRIGHT document, such as a solved problem or a theorem, and the query is a hard problem that this document helps to solve (Section 3.1). LeetCode and both TheoremQA domains define relevance the same way: a relevant document is a problem solved with the same algorithm or theorem, or the theorem itself [3]. Since base BGE-M3 also scored lower in these three domains than in any StackExchange domain, there was much room to improve, and every trained model scored above it in all three (Table 5.2).

Pony fell the most because its relevant pages differ from the positives of the Pony training examples. Each Pony query asks for a small program, such as one that checks whether a number is prime, and BRIGHT marks the same Pony tutorial pages on variables, operators, functions, and loops as relevant to almost every query [3]. In the training data, 2,192 of the 2,390 Pony examples instead pair a programming task with the documentation or source-code page of a specific part of the Pony standard library, such as a priority-queue task with the page of the `BinaryHeap` class. This is shown in Table 5.3, where every trained model found more relevant library pages than base BGE-M3, but fewer tutorial pages. Since 32 of the 43 relevant Pony pages are tutorial pages, this lowered the Pony recall of every trained model, although GRASS $\lambda = 0$ was affected the least.

AoPS also fell slightly for every trained model, and as with Pony, the training positives from its corpus are different documents from the ones BRIGHT marks as relevant. AoPS uses the same corpus of solved math problems as TheoremQA–Questions, but its queries are competition problems, and its relevant documents are other competition problems from contests such as the IMO and the AHSME or from the MATH dataset, that are solved with the same problem-solving technique [3]. The 9,614 training examples whose positives come from this corpus instead mostly use generated exercises from CAMEL, such as finding the frequencies of a sound signal with a Fourier transform, and multiple-choice algebra word problems from AQuA, and none of them use a document that is relevant for AoPS.

Outside Pony, the two methods that mine their own hard negatives, ANCE and GRASS, were the most robust, as ANCE and GRASS $\lambda = 0$ each fell below base BGE-M3 in six of the other eleven domains and GRASS $\lambda = 0.3$ and $\lambda = 0.5$ in only four, while in-batch and cross-batch fell below it in eight and seven domains and lost more recall in them (Table 5.2). This is likely because ANCE and GRASS train each query against a negative that the model being trained selects from the corpus or its cache and that is replaced as training goes on, whereas in-batch and cross-batch keep the fixed hard negative of each training example, which for the ReasonIR examples was written by an LLM. ANCE improved more than GRASS on both TheoremQA domains, probably because GRASS trained on half as many query examples (Table 5.7). This could not be tested separately, as GRASS also picks its queries and negatives differently from ANCE.

6.5. Limitations

Each method was trained only once, so differences in the non-statistically significant results could be due to small random variations. The same experiments should be run multiple times, and the results averaged to obtain more reliable estimates for each method.

GRASS had a smaller training budget than the other methods due to its query selection strategy. It should also be compared with a 1-epoch ANCE run. In addition, single-GPU tests should be run for both ANCE and GRASS to better assess the time savings from GRASS mining.

Uncertainty was only tested with three MC dropout passes, added to the similarity score with $\lambda = 0.3$ and $\lambda = 0.5$ (Table 5.6). This means that the negative result only holds for this setup, as other options, such as an ensemble of encoders or using uncertainty in the loss, were not tested. In addition, the reasons given in Section 6.2.2 for why uncertainty did not help are based on the selection diagnostics, as the training loss of the selected negatives was not measured.

The evaluation covers BRIGHT only, and since no trained model scored significantly above base BGE-M3 on Recall@1000 (Table 5.1), the results discussed focus more on comparing the training methods with each other. This is because BRIGHT is a dataset that struggles to perform well even for the 8-billion-parameter REASONIR model, so the small gains were to be expected. However, other datasets could have been explored, and smaller models should have been tested to verify whether the GRASS method can help fine-tune baseline models.

The MS MARCO experiment only checked the ANCE implementation and ran on less hardware than the original setup, with one GPU for training and one for mining instead of four each, and for 300,000 instead of 600,000 updates (Section 4.3.6). Its MRR@10 of 0.2699, versus the published 0.3300 (Section 5.6), suggests that the implementation works as intended, but it is not an exact reproduction. The experiments should run on more hardware resources to compare with the exact ANCE method.

6.6. Disclosure

I acknowledge the use of ChatGPT and Grammarly to assist in rephrasing certain sections of this thesis in line with TU Delft's publishing policies [25].

7

Conclusion

Dense retrievers are trained with a contrastive loss that compares the relevant document of each query against negative documents, and the model learns most from negatives that it scores close to the relevant one. In-batch training uses the documents of the other queries in the batch as negatives, which adds no extra cost, but these documents are mostly unrelated to the query and easy to tell apart from the relevant one. Larger batches add more of these negatives, which requires more GPU memory, while most of the added negatives remain easy. ANCE finds harder negatives by searching the whole corpus with the current model, which requires re-encoding and re-indexing the whole corpus at every refresh and makes training expensive on limited hardware. This thesis investigated whether such hard negatives can be found without re-encoding the whole corpus.

To this end, GRASS was developed. It mines hard negatives from a bounded cache of documents instead of the full corpus. The cache keeps documents that are selected as negatives, refreshes their representations as the model changes, and replaces unused documents with new candidates from the corpus. While one GPU trains the model, a second GPU mines new negatives from the cache, so training never waits for mining. Half of the queries in each mining round are replayed, with priority given to queries with a large gap between the scores of their negative and positive. GRASS can also add an MC dropout uncertainty term to the similarity score of each candidate. GRASS was compared with in-batch, cross-batch, and ANCE-style training, all fine-tuning BGE-M3 on the same training data and evaluated with Recall@1000 on the 12 BRIGHT domains.

For RQ1, GRASS showed that a dense retriever can be trained on hard negatives without large batches or global mining, as it trained with 64 queries per update and still reached a Recall@1000 on par with ANCE and significantly above cross-batch training, which used 1,024 queries per update (Section 6.2.1). In addition, GRASS used nearly all of its mined negatives while training on half as many query examples as ANCE, and outside Pony, it was, together with ANCE, the most stable of the trained models across the BRIGHT domains (Section 6.4). Adding the MC dropout uncertainty to the similarity score did change which negatives were selected, but it did not change Recall@1000 significantly compared with selecting by the similarity score alone (Section 6.2.2).

For RQ2, GRASS showed that hard negatives can be mined without having to re-encode the whole corpus while keeping Recall@1000 close to that of ANCE, as each GRASS mining round re-encodes only the cache documents that are refreshed or replaced. A GRASS mining round took approximately a third of the time of an ANCE mining round, and larger datasets or limited hardware would benefit from this way of fine-tuning.

This shows that, on BRIGHT, hard negatives can be mined from a cache of 32,000 documents that is updated during training, although no trained model scored significantly above base BGE-M3.

References

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

[2] Payal Bajaj et al. *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*. 2018. arXiv: 1611.09268 [cs.CL]. URL: https://arxiv.org/abs/1611.09268.

[3] Hongruixuan Chen et al. *BRIGHT: A globally distributed multimodal building damage assessment dataset with very-high-resolution for all-weather disaster response*. 2025. arXiv: 2501.06019 [cs.CV]. URL: https://arxiv.org/abs/2501.06019.

[4] Jianlv Chen et al. *M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*. 2025. arXiv: 2402.03216 [cs.CL]. URL: https://arxiv.org/abs/2402.03216.

[5] Daniel Cohen et al. "Not All Relevance Scores are Equal: Efficient Uncertainty and Calibration Modeling for Deep Retrieval Models". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2021, pp. 654–664. DOI: 10.1145/3404835.3462951. URL: http://dx.doi.org/10.1145/3404835.3462951.

[6] Matthijs Douze et al. *The Faiss library*. 2025. arXiv: 2401.08281 [cs.LG]. URL: https://arxiv.org/abs/2401.08281.

[7] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1050–1059. URL: https://proceedings.mlr.press/v48/gal16.html.

[8] Luyu Gao and Jamie Callan. *Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval*. 2021. arXiv: 2108.05540 [cs.IR]. URL: https://arxiv.org/abs/2108.05540.

[9] Luyu Gao et al. *Scaling Deep Contrastive Learning Batch Size under Memory Limited Setup*. 2021. arXiv: 2101.06983 [cs.LG]. URL: https://arxiv.org/abs/2101.06983.

[10] Luyu Gao et al. "Tevatron: An Efficient and Flexible Toolkit for Neural Retrieval". In: *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '23. Taipei, Taiwan: Association for Computing Machinery, 2023, pp. 3120–3124. ISBN: 9781450394086. DOI: 10.1145/3539618.3591805. URL: https://doi.org/10.1145/3539618.3591805.

[11] Kaiming He et al. *Momentum Contrast for Unsupervised Visual Representation Learning*. 2020. arXiv: 1911.05722 [cs.CV]. URL: https://arxiv.org/abs/1911.05722.

[12] Sebastian Hofstätter et al. *Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling*. 2021. arXiv: 2104.06967 [cs.IR]. URL: https://arxiv.org/abs/2104.06967.

[13] Jimmy Lin et al. *Pyserini: An Easy-to-Use Python Toolkit to Support Replicable IR Research with Sparse and Dense Representations*. 2021. arXiv: 2102.10073 [cs.IR]. URL: https://arxiv.org/abs/2102.10073.

[14] Erik M. Lindgren et al. "Efficient training of retrieval models using negative cache". In: *Proceedings of the 35th International Conference on Neural Information Processing Systems*. NIPS '21. Red Hook, NY, USA: Curran Associates Inc., 2021. ISBN: 9781713845393.

[15] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL]. URL: https://arxiv.org/abs/1907.11692.

[16] Ilya Loshchilov and Frank Hutter. *Online Batch Selection for Faster Training of Neural Networks*. 2016. arXiv: 1511.06343 [cs.LG]. URL: https://arxiv.org/abs/1511.06343.

[17] Sören Mindermann et al. *Prioritized Training on Points that are Learnable, Worth Learning, and Not Yet Learnt*. 2022. arXiv: 2206.07137 [cs.LG]. URL: https://arxiv.org/abs/2206.07137.

[18] Rodrigo Nogueira and Kyunghyun Cho. *Passage Re-ranking with BERT*. 2020. arXiv: 1901.04085 [cs.IR]. URL: https://arxiv.org/abs/1901.04085.

[19] Thang M. Pham et al. *SlimLM: An Efficient Small Language Model for On-Device Document Assistance*. 2024. arXiv: 2411.09944 [cs.CL]. URL: https://arxiv.org/abs/2411.09944.

[20] Yingqi Qu et al. *RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering*. 2021. arXiv: 2010.08191 [cs.CL]. URL: https://arxiv.org/abs/2010.08191.

[21] Thilina Chaturanga Rajapakse, Andrew Yates, and Maarten de Rijke. "Negative Sampling Techniques for Dense Passage Retrieval in a Multilingual Setting". In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '24. Washington DC, USA: Association for Computing Machinery, 2024, pp. 575–584. ISBN: 9798400704314. DOI: 10.1145/3626772.3657854. URL: https://doi.org/10.1145/3626772.3657854.

[22] Benjamin Reichman and Larry Heck. *Dense Passage Retrieval: Is it Retrieving?* 2024. arXiv: 2402.11035 [cs.CL]. URL: https://arxiv.org/abs/2402.11035.

[23] Tom Schaul et al. *Prioritized Experience Replay*. 2016. arXiv: 1511.05952 [cs.LG]. URL: https://arxiv.org/abs/1511.05952.

[24] Afrina Tabassum et al. *Hard Negative Sampling Strategies for Contrastive Representation Learning*. 2022. arXiv: 2206.01197 [cs.LG]. URL: https://arxiv.org/abs/2206.01197.

[25] TU Delft OPEN Publishing. *TU Delft OPEN Publishing policies*. 2026. URL: https://www.tudelft.nl/open-publishing/publish-with-us/resources-support/tu-delft-open-publishing-policies (visited on 11/17/2026).

[26] Christophe Van Gysel and Maarten de Rijke. "Pytrec$_eval : AnExtremelyFastPythonInterfacetotrec_eval$". In: *The 41st International ACM SIGIR Conference on Research amp; Development in Information Retrieval*. ACM, 2018, pp. 873–876. DOI: 10.1145/3209978.3210065. URL: http://dx.doi.org/10.1145/3209978.3210065.

[27] Xun Wang et al. *Cross-Batch Memory for Embedding Learning*. 2020. arXiv: 1912.06798 [cs.LG]. URL: https://arxiv.org/abs/1912.06798.

[28] Lee Xiong et al. *Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval*. 2020. arXiv: 2007.00808 [cs.IR]. URL: https://arxiv.org/abs/2007.00808.

[29] Jingtao Zhan et al. *Optimizing Dense Retrieval Model Training with Hard Negatives*. 2021. arXiv: 2104.08051 [cs.IR]. URL: https://arxiv.org/abs/2104.08051.

[30] Kun Zhou et al. *SimANS: Simple Ambiguous Negatives Sampling for Dense Text Retrieval*. 2022. arXiv: 2210.11773 [cs.CL]. URL: https://arxiv.org/abs/2210.11773.