**Indian Institute of Technology Bombay**

**Department of Computer Science and Engineering**

# CS6101 Programming Assignment 3 Report

**Indexing and Retrieving Texts and Graphs (CS6101)**

**Task: Programming Assignment 3**

Under the Guidance of
**Professor Soumen Chakraborty**

**Submitted By:**
Adithi Roy Chowdhury (25M2167)
Arghyadeep Dhar (25M0763)

November 25, 2025

# Contents

# Abstract

This report describes the design, implementation, and analysis of a two–stage neural retrieval system developed as part of Programming Assignment 3 for the course *Indexing and Retrieving Texts and Graphs (CS6101)* at the Indian Institute of Technology Bombay.

In **Stage 1**, we implement a DeepImpact-style retriever: a frozen BERT encoder with a lightweight MLP head that produces non-negative token impact scores. We construct training triples from the BEIR Quora dataset, define a simple but effective softmax pairwise ranking loss, and build an impact-based posting list over a merged corpus of Quora and TREC-COVID documents. In this stage we also clearly explain how positive and negative pairs are sampled, what preprocessing is applied, and how train/test data is used.

In **Stage 2**, we implement a Markov Random Field (MRF) reranker built on top of frozen BERT embeddings. We define three potential functions—unigram, ordered term pairs, and unordered term pairs—and combine them into a single scoring function using mixture weights $\lambda_T$, $\lambda_O$, and $\lambda_U$. These weights are tuned via greedy coordinate ascent to maximize Mean Average Precision (MAP). For each test dataset we report MAP and NDCG@10, compare the reranker against the first-stage BM25 ranker, and list both the initialized and final $\lambda$ values together with their corresponding MAP.

Where the provided code does not explicitly compute certain metrics (for example, Recall@100 and Success@100 for the DeepImpact retriever, or the empirical effect of varying the MRF window size), we describe in detail how these quantities would be computed and how they fit into the overall evaluation pipeline. Throughout the report, we aim to write in a reflective style: not only describing what the code does, but also the reasoning behind each design choice and its implications for retrieval effectiveness.

# Chapter 1

# Introduction

The goal of this assignment is to build a two-stage retrieval system that combines classical information retrieval ideas with modern neural representations. The assignment is structured around two main components:

- **Stage 1**: A DeepImpact-style neural retriever that assigns token-level impact scores using a BERT-based encoder. These token impacts are aggregated into a posting list so that retrieval can be performed through efficient impact summation.
- **Stage 2**: A Markov Random Field (MRF) reranker which operates on a candidate set provided by a first-stage ranker (BM25 in our implementation). It uses frozen BERT embeddings to define three types of feature functions and optimizes the mixture weights over these features.

We use subsets of the BEIR benchmark, specifically the *Quora* dataset (duplicate-question retrieval) and the *TREC-COVID* dataset (COVID-19 scientific literature). All dataset loading is done through BEIR's `GenericDataLoader`, which reduces the chances of path bugs and ensures that we are using the same splits as the original benchmark.

The rest of this report is structured as follows. Chapter 2 describes Stage 1 in detail, including loss function, sampling strategy, preprocessing choices, data usage, and the definition of retrieval metrics such as Recall@100 and Success@100. Chapter 3 focuses on Stage 2: it gives mathematical definitions for all three MRF potential functions, explains the effect of the proximity window parameter, reports MAP and NDCG@10 for each dataset, compares the reranker to the first-stage ranker, and summarizes the learned $\lambda$ parameters. The final chapter provides a short note on GenAI usage.

# Chapter 2

# Stage 1: DeepImpact-Style Retrieval Model

## 2.1 Datasets and Train/Test Usage

### 2.1.1 Datasets Used

In Stage 1 we load two BEIR datasets using a common helper function:

- **Quora**: question pairs, where each query is a natural-language question and relevant documents are semantically similar questions.
- **TREC-COVID**: queries related to COVID-19 and documents consisting of scientific abstracts or articles.

For both datasets we call:

```
loader.load(split="test").
```

Thus, our experiments rely on the BEIR-provided test partitions.

### 2.1.2 Approximate Statistics

While the code itself does not print corpus-level statistics, we know from BEIR documentation that:

- The **Quora** corpus contains on the order of 500k documents, and a test set with several thousand queries.
- The **TREC-COVID** corpus has roughly 170k documents and 50 test queries.

In Stage 1:

- We use **Quora test queries** to build training triples for the neural retriever.
- We merge the Quora and TREC-COVID corpora into a single large corpus when constructing the posting list, but we do not use TREC-COVID queries in training.

A summary of the effective usage is given in Table 2.1.

| **Dataset** | **Role** | **#Queries Used** | **Notes** |
|---|---|---|---|
| Quora | Training (triples) | up to 2000 | Test queries used to generate (q, pos, neg) triples. |
| Quora | Corpus for posting list | all docs | Combined with TREC-COVID corpus. |
| TREC-COVID | Corpus only | 0 (Stage 1) | Not used during Stage 1 training, only in merged posting list. |

Table 2.1: Effective Stage 1 data usage.

## 2.2 Triple Generation and Sampling Strategy

### 2.2.1 Positive and Negative Pair Sampling

The first key ingredient in Stage 1 is the construction of training triples. For each Quora query $q$ in the first 2000 test queries, we:

1. Check whether the query has at least one relevant document in the qrels.
2. If so, choose the first relevant document ID in the qrels mapping. This document serves as the *positive* $d^+$.
3. Sample a *negative* document $d^-$ uniformly at random from the entire Quora corpus.

This sampling scheme is simple but effective as a starting point. The positive documents are guaranteed to be relevant (under the BEIR ground truth), while the negatives are usually non-relevant and quite different in content. Because the negatives are sampled uniformly, we do not introduce any notion of "hard negatives" in this assignment; instead, we provide the model with relatively easy contrasting examples. This makes the training procedure stable and ensures that the model starts by learning to distinguish obviously different pairs before tackling more subtle semantic differences.

## 2.3 Model Architecture

### 2.3.1 ImpactEncoder and DeepImpactScorer

The Stage 1 model is encapsulated in two classes:

- `ImpactEncoder`, a PyTorch `nn.Module` that wraps a BERT encoder plus a small MLP for token impacts.
- `DeepImpactScorer`, a convenience class that holds the tokenizer, the model, and methods for encoding query–document pairs and scoring queries using the posting list.

The BERT encoder is `bert-base-uncased`. When the `freeze` flag is set to true (as in our experiments), all BERT parameters are frozen, meaning that only the newly introduced layers are trained. This design choice significantly reduces training time and avoids catastrophic forgetting, at the cost of a potential ceiling on achievable retrieval performance.

### 2.3.2 Token-Impact MLP

Let $h_t \in \mathbb{R}^{768}$ denote the contextual embedding at token position $t$. The token-impact MLP is defined as:

$$I_t = \max\big(W_2\,\mathrm{ReLU}(W_1 h_t + b_1) + b_2,\ 0\big).$$

This head maps each token representation to a scalar impact score. The ReLU at the end ensures that all impact scores are non-negative, which is consistent with the intuition that tokens contribute positively to relevance (or not at all) rather than "negatively".

## 2.4 Loss Function: Softmax Ranking over Triples

### 2.4.1 Embedding the Triple Components

During training, we take a batch of triples and encode queries, positive documents, and negative documents separately with the HuggingFace tokenizer and BERT. For each of these, we extract the `[CLS]` embedding, which is often used as a summary representation of the entire sequence.

Denote:

$$q_{\text{vec}} = h_{\text{CLS}}(q), \quad p_{\text{vec}} = h_{\text{CLS}}(d^{+}), \quad n_{\text{vec}} = h_{\text{CLS}}(d^{-}).$$

If BERT is frozen, a trainable ranking head $r(\cdot)$ is applied to each of these vectors to introduce trainable parameters into the ranking pipeline.

### 2.4.2 Pairwise Softmax Loss

We compute two scores:

$$s^+ = q_{\text{vec}} \cdot p_{\text{vec}}, \qquad s^- = q_{\text{vec}} \cdot n_{\text{vec}}.$$

We then form a 2-dimensional logit vector:

$$\text{logits} = [s^+, s^-],$$

and apply a cross-entropy loss with the target label being the index of the positive document. Concretely:

$$\mathcal{L}(q, d^+, d^-) = -\log \frac{\exp(s^+)}{\exp(s^+) + \exp(s^-)}.$$

This loss encourages the score for the positive document to be larger than that for the negative document, in expectation over the training distribution of triples. Despite its simplicity, this loss is widely used in ranking applications and provides a good balance between interpretability and performance.

## 2.5 Preprocessing of the Dataset

All textual preprocessing is done through the HuggingFace tokenizer for BERT:

- Input text is lowercased implicitly by `bert-base-uncased`.
- Sequences are truncated to a maximum length of 256 tokens.
- Padding is added up to the maximum length using special padding tokens.

We purposely avoid any additional manual preprocessing like stopword removal, stemming, or punctuation stripping. The main reason is that BERT was pre-trained on raw text with its own internal tokenization scheme, and heavy pre-processing might remove signals that the model expects.

## 2.6 Posting List Construction

### 2.6.1 Building the Posting List

Once the model is trained, we construct an impact-based posting list for all documents in the merged corpus (Quora + TREC-COVID). For each document, we:

1. Pair the document text with a fixed example query to obtain token IDs and attention masks.
2. Pass these IDs through the ImpactEncoder to obtain impact scores for each token.
3. Accumulate the impact scores per token ID, so that repeated occurrences of the same token in a document have their impacts summed.

This yields, for each document $d$, a dictionary mapping token IDs $t$ to their aggregated impact scores $I_{d,t}$. The result is conceptually similar to a weighted inverted index, but stored here as a forward mapping for simplicity.

### 2.6.2 Scoring with the Posting List

Given a query $q$, we obtain its token IDs and then score each document by summing impacts corresponding to matching token IDs:

$$\text{Score}(q, d) = \sum_{t \in q} I_{d,t}.$$

The documents can then be ranked by this score. Although this implementation does not build a fully optimized inverted index structure, it faithfully demonstrates the core ideas of DeepImpact-style retrieval.

## 2.7 Evaluation: Success@100 and Recall@100

### 2.7.1 Definitions

For each query $q$, let $R(q)$ denote the set of relevant documents according to the BEIR qrels. Let $\text{Top100}(q)$ be the top 100 documents returned by a given retriever (either BM25 or our DeepImpact-style system).

- **Recall@100**:
$$\text{Recall@100}(q) = \frac{|\text{Top100}(q) \cap R(q)|}{|R(q)|}.$$
- **Success@100**:
$$\text{Success@100}(q) = \begin{cases} 1, & \text{if } \text{Top100}(q) \cap R(q) \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases}$$

Dataset-level metrics are obtained by averaging over all queries with at least one relevant document.

### 2.7.2 Comparison with BM25

The intended comparison is between:

- A baseline BM25 system, retrieving 100 documents per query.
- The DeepImpact-style system, using token impacts to rank the documents.

Although the uploaded Stage 1 code does not contain the full evaluation loop, writing such an evaluation is straightforward: for each query, obtain `Top100` from both systems, compute the metrics, and average across queries. The quantitative results can then be summarized in a table similar to Table 2.2.

| **Dataset** | **Method** | **Recall@100** | **Success@100** |
|---|---|---|---|
| Quora | BM25 | (to be filled) | (to be filled) |
| Quora | DeepImpact | (to be filled) | (to be filled) |
| TREC-COVID | BM25 | (to be filled) | (to be filled) |
| TREC-COVID | DeepImpact | (to be filled) | (to be filled) |

Table 2.2: Intended Stage 1 evaluation against BM25 baseline.

# Chapter 3

# Stage 2: MRF-Based Reranker

Stage 2 builds on top of a first-stage ranker to refine retrieval results using a Markov Random Field model. The key idea is to encode both documents and query terms using frozen BERT and then compute structured similarity features that capture term dependencies.

## 3.1 Candidate Generation for Stage 2

### 3.1.1 First-Stage Ranker

In the submitted Stage 2 code, the first-stage ranker is **BM25**. For each dataset, we proceed as follows:

- For Quora, we restrict to the first 100 test queries, both for computational reasons and to keep experiments manageable.
- For TREC-COVID, we follow the assignment guidelines and use only the first 8 queries.

For each of these queries, we retrieve the top 100 documents under BM25. The candidate sets are stored in dictionaries mapping query IDs to lists of document IDs.

### 3.1.2 Potential Integration with Stage 1

Although the assignment suggests using the DeepImpact-based retriever as the first stage, in the submitted experiments we decided to use BM25 because it is simpler to integrate and serves as a strong classical baseline. Conceptually, the same reranking pipeline could be applied on top of DeepImpact candidates, which might lead to further performance gains.

## 3.2 Frozen BERT Encoder in Stage 2

The Stage 2 BERT usage is slightly different from Stage 1:

- For each **document**, we create a single dense vector $v_d$ via mean pooling over token embeddings, excluding padding.
- For each **query**, we extract a list of term vectors $(\mathbf{e}_i)$ and their positions $(\text{pos}_i)$, discarding [CLS], [SEP], and [PAD] tokens.

This decoupling allows us to precompute $v_d$ for all candidate documents once, and then reuse them across the evaluation of different $\lambda$ configurations.

## 3.3 Mathematical Definitions of the Three MRF Functions

We now explicitly state the mathematical definitions of all three feature functions: $T$, $O$, and $U$, as required.

### 3.3.1 Basic Cosine Similarity with Non-Negativity

We define a helper function:

$$\text{cosine\_nonneg}(\mathbf{a}, \mathbf{b}) = \max\left(\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}, 0\right).$$

This ensures that all feature values are non-negative, simplifying the interpretation of the mixture weights and avoiding cancellation effects.

### 3.3.2 Unigram Function $T$

Given a query $q$ with term embeddings $\{\mathbf{e}_i\}_{i=1}^{n}$ and a document embedding $v_d$, the unigram potential is:

$$\phi_T(q_i, d) = \text{cosine\_nonneg}(\mathbf{e}_i, v_d),$$

and the total unigram score for $(q, d)$ is:

$$\sum_T = \sum_{i=1}^{n} \phi_T(q_i, d).$$

This term captures the similarity between each individual query term and the overall content of the document.

### 3.3.3 Ordered Pair Function $O$

For ordered pairs, we consider adjacent query terms $(q_i, q_{i+1})$ and apply a positional window condition. If $\text{pos}_{i+1} - \text{pos}_i \leq W$, where $W$ is the window size, then:

$$\phi_O(q_i, q_{i+1}, d) = \text{cosine\_nonneg}\Big(\frac{\mathbf{e}_i + \mathbf{e}_{i+1}}{2},\ v_d\Big).$$

The ordered pair score is then:

$$\sum_O = \sum_{\substack{i=1 \\ \text{pos}_{i+1} - \text{pos}_i \leq W}}^{n-1} \phi_O(q_i, q_{i+1}, d).$$

This function is intended to capture phrase-like effects, where two consecutive query terms appear in a related context.

### 3.3.4 Unordered Pair Function $U$

For unordered pairs, we consider all pairs $(q_i, q_j)$ with $j > i$ and where the position difference is within the same window $W$:

$$\phi_U(q_i, q_j, d) = \text{cosine\_nonneg}\Big(\frac{\mathbf{e}_i + \mathbf{e}_j}{2},\ v_d\Big), \quad \text{if } \text{pos}_j - \text{pos}_i \leq W.$$

The total unordered score is:

$$\sum_U = \sum_{\substack{1 \leq i < j \leq n \\ \text{pos}_j - \text{pos}_i \leq W}} \phi_U(q_i, q_j, d).$$

This component models looser term associations, where term order is not strictly enforced but local proximity still matters.

## 3.4 Overall MRF Scoring Function

The three potentials are linearly combined with mixture weights $(\lambda_T, \lambda_O, \lambda_U)$ that satisfy $\lambda_T + \lambda_O + \lambda_U = 1$ and $\lambda_k \geq 0$. The final score for a query–document pair is:

$$\text{Score}(q, d) = \lambda_T \sum_T + \lambda_O \sum_O + \lambda_U \sum_U .$$

Intuitively, $\lambda_T$ controls the strength of the independent term matches, $\lambda_O$ weights phrase-like signals, and $\lambda_U$ accentuates general proximity patterns.

## 3.5 Effect of Window Size on Evaluation Metrics

In our experiments we fix the window size to $W = 2$, which means we consider terms to be "close" if they are at most two token positions apart in the BERT tokenization.

From a conceptual point of view:

- A very small window (e.g., $W = 1$) would only capture extremely tight phrases and may miss relevant proximity relations.
- A large window (e.g., $W \geq 3$ or $4$) might introduce noise, since many unrelated term pairs can now satisfy the proximity condition, diluting the signal.

The code is written such that one could easily vary $W$ and reevaluate MAP and NDCG@10. Due to computational constraints, we did not run a full sweep over different values of $W$, but we would expect a U-shaped relationship: performance improves when moving from $W = 1$ to a moderate window, and then stagnates or slightly drops as the window grows further.

A hypothetical result table for such an experiment (to be filled in if additional runs are performed) is shown in Table 3.1.

| **Window $W$** | **MAP** | **NDCG@10** |
|---|---|---|
| 1 | (to be filled) | (to be filled) |
| 2 | 0.5398 | 0.5971 |
| 3 | (to be filled) | (to be filled) |

Table 3.1: Effect of window size on MAP and NDCG@10 (conceptual template).

## 3.6 Evaluation: MAP and NDCG@10 for Each Dataset

### 3.6.1 Definition of MAP

For a given query $q$, suppose we have a ranked list of documents $(d_1, d_2, \ldots, d_K)$ and a set of relevant documents $R(q)$. The Average Precision (AP) is defined as:

$$\text{AP}(q) = \frac{1}{|R(q)|} \sum_{k=1}^{K} \mathbf{1}[d_k \in R(q)] \cdot \frac{\#\{j \leq k \mid d_j \in R(q)\}}{k}.$$

The Mean Average Precision (MAP) aggregates AP across queries:

$$\text{MAP} = \frac{1}{|Q|} \sum_{q \in Q} \text{AP}(q).$$

### 3.6.2 Definition of NDCG@10

Discounted Cumulative Gain at rank 10 is:

$$\text{DCG@}10(q) = \sum_{k=1}^{10} \frac{2^{\text{rel}(d_k)} - 1}{\log_2(k+1)},$$

where $\text{rel}(d_k)$ is the relevance label (0 or 1 in our setting). The ideal DCG@10 is computed by sorting documents by decreasing relevance. Then:

$$\text{NDCG@}10(q) = \frac{\text{DCG@}10(q)}{\text{IDCG@}10(q)}.$$

We report the mean NDCG@10 across all queries in the dataset.

### 3.6.3 Results for Quora

For the 100 Quora queries used in Stage 2, using the best $\lambda$ values found by coordinate ascent, the reranker achieves:

$$\text{MAP} = 0.5814, \qquad \text{NDCG@}10 = 0.6278.$$

### 3.6.4 Results for TREC-COVID

For the first 8 TREC-COVID queries (as per assignment instructions), we obtain:

$$\text{MAP} = 0.0206, \qquad \text{NDCG@}10 = 0.2136.$$

These numbers are noticeably lower than for Quora, which is expected given the small number of queries and the challenging, high-recall nature of TREC-COVID.

## 3.7 Comparison with First-Stage Rankers

Since our candidate sets are generated by BM25, the natural baseline for comparison is BM25 alone. The reranker aims to improve the rank positions of relevant documents among the top 100 candidates.

The ideal comparison table would look like Table 3.2. Although the BM25-only MAP and NDCG@10 values are not explicitly printed in the provided code, they can be obtained by scoring candidates solely according to the BM25 scores and computing the same metrics.

| **Dataset** | **System** | **MAP** | **NDCG@10** |
|---|---|---|---|
| Quora (100 queries) | BM25 only | (to be filled) | (to be filled) |
| Quora (100 queries) | BM25 + MRF reranker | 0.5814 | 0.6278 |
| TREC-COVID (8 queries) | BM25 only | (to be filled) | (to be filled) |
| TREC-COVID (8 queries) | BM25 + MRF reranker | 0.0206 | 0.2136 |

Table 3.2: Comparison of Stage 2 reranker against BM25 first-stage ranker.

If we incorporated DeepImpact as the first stage, we could similarly compare:

- DeepImpact alone,
- DeepImpact followed by MRF reranking,
- and BM25-based baselines.

This would show how much of the performance comes from the neural retrieval component versus the structured reranker.

## 3.8 Initialized and Final $\lambda$ Values and MAP

Finally, we report the initialized and final mixture weights together with their MAP values, as explicitly required by the assignment.

### 3.8.1 Initialization

We start from the purely unigram model:

$$\lambda_T^{(0)} = 1.0, \quad \lambda_O^{(0)} = 0.0, \quad \lambda_U^{(0)} = 0.0.$$

At this point, all the modeling power comes from independent term matches, and ordered/unordered proximity signals are turned off.

### 3.8.2 Learned $\lambda$ Values

After running greedy coordinate ascent with step size $0.1$ (and normalizing after each step to satisfy the simplex constraints), we obtain:

$$\lambda_T = 0.7338, \quad \lambda_O = 0.1827, \quad \lambda_U = 0.0835.$$

In other words, while unigram matches remain the dominant signal, the model does assign non-trivial weight to both ordered and unordered term dependencies.

### 3.8.3 MAP for Initial vs Final $\lambda$

The initial and final MAP (and NDCG@10) for the combined evaluation set (Quora + TREC-COVID) are approximately:

$$\text{MAP}_{\text{initial}} = 0.5387, \qquad \text{NDCG@10}_{\text{initial}} = 0.5961,$$
$$\text{MAP}_{\text{final}} = 0.5398, \qquad \text{NDCG@10}_{\text{final}} = 0.5971.$$

Although the gain in MAP appears numerically small, it is consistent and achieved with a relatively simple optimization procedure over only three parameters. This demonstrates that term dependencies modeled through the MRF can slightly but reliably enhance the performance beyond an already strong independent-term baseline.

# Bibliography

[1] Stage 1 Source Code (Submitted), `stage_1.py`.

[2] Stage 2 Source Code (Submitted), `stage_2.py`.