Elahe Hajihashemi Varnousfaderani

Challenges and Insights in Semantic Search Using Language Models

Master’s thesis of the Faculty of Information Technology

November 5, 2023

University of Jyväskylä
Faculty of Information Technology

Author: Elahe Hajihashemi Varnousfaderani

Contact information: elahe.hajihashemi@gmail.com

Supervisors: Dr. Oleksiy Khriyenko

Title: Challenges and Insights in Semantic Search Using Language Models

Työn nimi: haasteita ja oivalluksia semanttisessa haussa kielimalleja käytettäessä

Project: Master's thesis in Cognitive Computing and Collective Intelligence

Study line: Information Technology

Page count: 148 (122 = page count without appendices; 26 = page count of appendices)

Abstract

Information Retrieval systems such as search engines, originally designed to assist users in finding information, have evolved to become more potent and have found utility in wider range of applications by incorporating contextual comprehension using Language Models. Selecting the proper Language Model corresponding to the desired task is a challenging multi-objectives problem as each model has specific set of attributes which affect the performance. Accuracy, resource and time consumption are the most important objectives considered in assessing the quality of a search system. These objectives are addressed in this research by exploring the performance of two Language Models with variant characteristics in developing a semantic search pipeline. The studied Language Models include a distilled version of BERT model fine-tuned on specific task and GPT-2 as a general pre-trained model with huge number of parameters.
The semantic search pipeline consisting of mapping the contents and queries into a common vector space using Large Language Model and finding the most relevant results is implemented in this study as experimental set up of the qualitative research. Utilizing evaluation metrics to assess the model's performance necessitates the availability of ground truth data. Therefore, current research brings up various approaches aimed at generating synthetic ground truth to tackle evaluation and fine-tuning challenges when labeled data is scarce. To follow the research objectives, quantitative

data is gathered through an experimental setting and conclusions are drawn and recommendations are raised by analyzing the results of the experiments.

The experimental results indicate the size of the model should not be the major criterion in selecting the language model for downstream tasks. The model architecture and being fine-tuned on special dataset will dramatically affect the performance as well. As it is shown by results, the smaller fine-tuned model for semantic textual similarity surpasses the larger general model. The experiment on investigating the proposed approaches for generating annotations signifies that those methods are decently applicable in computing evaluation metrics and can be extended to fine-tuning.

The results demonstrate that the task-oriented transferred learning by distillation and fine-tuning can compromise the learning capacity instilled in general models by a larger number of parameters, but it should be investigated in future research regarding the values set to various variables in this research e.g., the number of tokens considered in splitting the large text into smaller chunks. Moreover, it would be worthful to fine-tune the general large model as well in the future to compare them in a more comparable condition.

Keywords: Natural Language Processing, Semantic Search, Large Language Models, Generative Models, Fine-tuning, Transfer Learning

List of Figures

List of Tables:

Contents

1. Introduction

Nowadays search applications has taken a crucial role in various scenarios from daily life to industrial and medical use cases. Users expect to interact with a system that provide the most relevant results fast, even they would be pleased to communicate with their documents in natural language. In every organization and even in every personal system there are many files that we need to look through them frequently to find the required information. In such situations, the presence of an application that can receive the requested information in natural language and be able to search on behalf of the requester then provide them with the accurate results or exact answer based on the retrieved sources is much appreciated. It can save a lot of time in any kind and any size of organization and increase the productivity of the operators as they can invest that time to do more meaningful tasks and enhance their creativity. This kind of search applications can perform as operational assistants. For example, an ERP (Enterprise Resource Planning) company could integrate a search application to their system to help their customers to get insights from employee feedback using semantic search. Another case could be equipping the manual handbooks of the products with a semantic search to look for the information by asking natural language questions and get the exact answer quickly instead of advocating some time to go through all pages of the handbook. Imagine, how helpful it could be for an online learning company to boost their core search and question-answering mechanism with an AI-powered semantic search to support the learners to find real-time answers to their questions based on the taught materials in any modality from text to image and video.

It worth to note that search engines and recommendation systems are different types of Information Retrieval systems that can benefit from a deep understanding of the domain, resulting in a semantic search engine. For instance, a movie recommender system or an item recommender integrated with an online fashion shop are two examples of the recommendation engines that employ the search application to find the similarities in the recent request, historical data, and trend patterns.

The main objective of the search assistance is to facilitate accessibility to information hidden in a huge number of documents within different sources, structures and hierarchies while being served with low latency and high accuracy considered as user satisfaction factors. Besides including user satisfaction factors, delivering a solution with the lower carbon footprint should be considered as a pivotal key. This research is an attempt to introduce AI-powered search as an approach in developing semantic search applications while considering the most effective elements in determining the quality and speed of performance besides keeping the carbon footprint at the lowest level. The main point here is that these three objectives may conflict, and the owner of the solution should more precisely consider them in the development process to make the optimal decision. So, this research is going to examine how some required considerations influence the performance as a measure in making decision on the mentioned objectives. For instance, the size of the utilized Language Model in development of the AI-powered search, the effect of applying a task-specific or generally trained model and re-ranking the search results with machine learning approaches. It should be mentioned that for comparing the influence of the factors, a proper judgement list is required even though it is usually not feasible in real-world use cases. This is another aspect that this research addresses.

Traditional approaches in Natural Language Processing utilized in search application development primarily rely on keyword-based indexing and lexical matching consequently, they do not deeply consider the context or meaning of the query and contents. To create a more intelligent solution that can grasp search queries and offer responses grounded in semantically related information from the knowledge source, it is essential to leverage Language Model and Transfer Learning. This approach goes beyond conventional keyword matching, striving to comprehend the underlying meaning and context of user queries. The result is a delivery of more precise and contextually relevant search results.

The semantic search pipeline involves two key phases. In the first phase, the input query and the collection of contents are converted into numerical representations. In the second phase, the most similar representations to the query are sought, and the best-matching original content returns. The magic happens when the more advanced models are applied to capture and store more meaningful context from the query and contents in the form of the numerical representations that are called word embeddings. These embeddings represent the semantic meaning of passages in a dense vector space, where similar contents are located close together. The retrieved results would be ranked based on the level of the measured similarity. As a further step, the basic retrieved contents could be re-ranked based on more advanced metrics to achieve better performance.

In the literature there has been several methods proposed to score the contextual relativeness in text-based information. Reimers et al. (2019) utilized an approach called Cross-Encoder with BERT language model to predict a target value which is the textual similarity of the inputs. In this method, two sentences are fed to the Transformer network which is followed by a classifier to define the score of the similarity between them in the terms of the probability of being similar or not. The Cross-Encoder utilized for data annotation as a technique to refine the human annotated training data for Bi-Encoder models (Chiu et al. 2022).

Zhang et al. (2020) proposed another method, called BERTScore, as an automatic evaluation metric for text generation. BERTScore would be fed with two sentences as inputs then it computes the similarity score between reference and candidate sentences in a token-wise level. The token similarity is calculated using contextual embedding extracted by a Language Model. They claimed that BERTScore correlates well with human judgment and could be a strong metric to measure the model selection performance (Zhang et al. 2020). In this thesis, Cross-Encoder and BERTScore methods are applied in the pipeline to examine their effect on the overall performance.

There are various Language Models (LM) with a diversity of characteristics that can be utilized in textual information retrieval. The number of parameters that the model has and the number of tokens in the input that the model can handle are some of those attributes that would affect the user satisfaction from two perspectives including accuracy and latency. Therefore, this research explores the performance of various models regarding these perspectives in order to select the better model to satisfy the user's expectations in a specific downstream task besides considering the sustainability concept. In other words, it is an attempt to see if there is any room to deliver a satisfactory solution to the use case with minimal sufficient resource consumption.

In the recent years of Deep Learning fast growth, training a variety of Language Models resulted in a mutation in the Natural Language Processing applications. Devlin et al. (May 2019) introduced a language representation model called BERT. BERT is a Bidirectional Encoder Representation from Transformers. It is a pre-trained model that can be fine-tuned for any specific downstream task. It creates decent results without any need for task-specific architecture modification. As one of the most famous Transformer models the *BERT* pre-trained on a large corpus of unlabeled text, it generates contextually rich embeddings for words and sentences. *BERT* has achieved state-of-the-art performance in various NLP tasks (Devlin et al. 2019).

Sentence-BERT was proposed in August 2019 by Reimers et al. which is a modification of BERT model. Even though BERT achieved excellent results on sentence-pair regression tasks like semantic textual similarity, it is not computationally effective because a huge number of combinations of request and contents' sentences need to be fed to the model. Therefore, Sentence-

BERT emerged to derive semantically meaningful sentence embeddings that can be compared simply in a vector space. This model reduced the consumed time for finding the most similar pair dramatically while maintaining the accuracy (Reimers et al. 2019).

If an NLP task is developed in a response to a use-case that is required to be operated on the edge, it will be more constrained regarding computational and memory resources. In March 2020, DistilBERT was proposed as a method to pre-train a smaller general-purpose language representation model which can be fine-tuned to achieve a good performance on different tasks. Despite other distilled models, knowledge distillation applied in the pre-training phase. It reduced the size of the BERT model while retaining its language understanding capabilities besides faster operation which makes it suitable for operating on edge (Sanh et al. 2020).

Due to the scarcity of labeled data for training large language models for specific NLP tasks like question-answering, semantic similarity assessment, and textual entailment, it would be challenging to train an adequate discriminative task-specific model. Therefore, Radford et al. (2018) proposed an approach for generative pre-training of a model on a diverse corpus of unlabeled text, followed by discriminative fine-tuning on each specific task which is the fundamental of the GPT (Generative Pre-trained Transformer) models.

GPT-2 Large is the second version of GPT models that is specific regarding introducing a language model that can learn the task besides the language without any explicit supervision. It is a general-purpose language model that stands out for its extensive model size and numerous parameters, enabling it to comprehend complex patterns and dependencies in natural language in addition to the success of zero-shot task transfer. However, due to its large size and complexity, it demands significant computational resources for both training and inference (Radford et al. 2019).

Verma (2021) utilized the Sentence-BERT to implement a semantic search to improve the search accuracy by understanding the content of the search query. It is mentioned that selecting the appropriate model for your specific task remains the most important and the choice should be based on the task requirements. The Synthetic Query Generation technique is applied by him to tackle the challenge of the label scarcity to fine-tune the model (Verma, 2021). The outcomes of applying re-ranker in this research is compared with the fine-tuned results of Verma's implementation.

Choosing the right language model for a specific task is critical and challenging. Therefore, in this research two language models with different features are selected to be compared to address the influential factors in making the decision on the fitting model for developing solution for a specific task. GPT2 is selected as a general-purpose model with a huge number of parameters with wide capacity of knowledge capturing and the other one is a small, distilled version of BERT which is specifically fine-tuned for semantic text similarity task.

Assessing the performance of an Information Retrieval (IR) system involves evaluating how effectively the system can retrieve relevant information in response to user queries. The performance evaluation provides a measure for system developers to assess the effectiveness of the applied pipeline, ultimately leading to improved search results and increased user satisfaction. The size of the model, which is determined by the number of trained parameters in the training phase, is a crucial factor in model performance as the bigger model has the more capacity of contextual learning. On the other hand, it comes with the more resource consumption for retraining and serving which would lead into latency and higher carbon footprint in the solution. Therefore, it is essential to computes the model performance besides applying techniques that can improve the model performance in a lower cost. Re-ranking is one those kinds of techniques that can be investigate if in combination with a smaller model can lead into better performance.

To assess the performance of the model, the predicted relevant items should be compared to the ground truth. Consequently, proper ground truth is essential for evaluation. However, it is not available in many real-world situations. Ground truth, which is also called judgment list in this research, is a list of relevant items among the source of knowledge with respect to each query. In addition to evaluation, the ground truth is required in fine-tuning the pre-trained model on the domain-specific data.

Because of the importance of the presence of the judgment list, this research aims to explore various methods to create synthetic annotations using Cross-Encoder and BERTScore to address the challenge of limited ground truth, which is crucial for both evaluation and fine-tuning processes.

In total the current research is an experimental attempt in answering the following research questions:

1. How to create a semantic search using language models?
2. How to tackle the difficulty of fine-tuning a language model when there is an absence of labeled data?

3. Utilizing MLOps framework to develop and deliver an end-to-end semantic search solution.
4. Exploring whether a general larger language model outperforms a task-specifically fine-tuned smaller one in semantic search.
5. Investigating the performance of models with varying numbers of parameters in the field of natural language understanding.

The hypothesis in this research is that if a general larger model with higher contextual capacity would be able to perform comparably to a smaller one with fewer parameters which is fine-tuned. However, there are various items that should be considered in drawing a conclusion, for instance, task conditioning in the pre-training stage, task-oriented layer added on top of the Transformer architecture and being fine-tuned with specific small dataset. This research is organized to investigate the hypotheses by exploring the effect of the utilized model in a conductive research method supported by analyzing the quantitative data collected in an experimental setting.

There are two typical evaluation approaches in Information Retrieval based on user involvement. If real users are asked to judge search engine performance, especially in Interactive Information Retrieval, user feedback, and user interaction are crucial to evaluate. On the contrary, in automatic evaluation, a data corpus and binary evaluations are required for performance evaluation (Pérez-Agüera et al., 2010).

Evaluation based on test collection is a widely utilized method for developing retrieval strategies. Benchmarks provide a standardized and replicable experimental setup, allowing multiple researchers to evaluate and compare results consistently. While user-oriented evaluation is valuable, it is costly, complex, and challenging to replicate. The stability and standardization offered by test collections make it an appealing choice for evaluation purposes (Clough & Sanderson, 2013).

A test collection typically comprises three main components: a document collection, a set of topics representing users' information needs, and relevance judgments indicating which documents in the collection are relevant to each topic (Clough & Sanderson, 2013).

To establish an evaluation based on test collection to develop an automatic evaluation process, a collection of documents supplied with metadata and a set of queries are needed. Therefore, a publicly available dataset inspiring for Information Retrieval is used as the source of information besides a set of designed queries, to nourish the experimental settings to collect numerical data on performance evaluation to accomplish primary research data.

There are two substantial concepts in the performance evaluation of the semantic search pipeline and IR systems encompass efficiency and effectiveness. Efficiency is a measure of the number of required resources to achieve the search results e.g., computational resources in addition to low latency. While effectiveness implies the quality of the returned results which can be measured with the evaluation metrics as an automatic approach or by user involvement through their interactions and feedback (Clough & Sanderson, 2013).

This research attempts to simulate a real problem situation of evaluating the performance of a semantic search system when there is no label for the documents in the corpus. It is highly dealt with bringing up methods to calculate the numerical metrics and analysis them so, it is a well-suited match for quantitative research.

An experimental evaluation is done based on different metrics to distinguish different results returned using two samples of language models. The goal is to compare the performance of the large but general-purpose language model, GPT-2, and a small but fine-tuned version of BERT model. Therefore, how changing the language model, as the prominence factor in the pipeline, may affect effectiveness and efficiency which is represented by the evaluation metrics and time consumption respectively is investigated. Consequently, the conduction of this research is implemented in correlational and experimental setups.

In the current thesis a dataset is chosen as a test collection that contains a document collection, topics, and complementary information about documents but there is no relevance judgment. Accordingly, the Wikipedia Movie Plots is selected as dataset in this research which is described in detail in sections 3.3.1 and 3.3.2. Besides that, a set of queries is required to be delivered to the semantic search procedure and produce results to be studied. The queries fed to the pipeline can be as simple as multiple keywords or in the form of a natural language query that conveys more contextual information. The experimental setting of this thesis is supplied with NLP queries designed to cover various topics in the document collection.

Numerical data is collected through implementing three methods suggested to provide a relevance judgment list and these qualitative data is used for analyzing the outcomes in descriptive statistics manner.

Effectiveness is evaluated in this thesis using neural-based evaluation metrics like Bi-Encoder, Cross-Encoder, and BERTScore regarding several proposed techniques to approximate the human judgment to tackle the lack of annotation in the dataset. Precision, recall, and Mean Reciprocal Mean (MRR) are used as statistics to compare the evaluation metrics and analysis the values of those metrics for different queries.

Quantitative research is the best match for the research targets in this thesis because the experimental setting can be replicated according to the implementation details besides the results can be compared statistically and directly. Moreover, the procedure is reliable and consistent so large samples can be processed and analyzed. Therefore, the hypothesis could be established and examined carefully before concluding.

The analytics phase in conducting the research goals which is done in the quantitative method is supported by scrutinizing the results that are provided by the search system before being mapped to the numbers which could be considered as qualitative observations. In this regard, the researcher is in the role of the decision maker to review the relevancy level of the results.

The first research question explores developing of a search application using Large Language Models. Then how to conquer the challenge of fine-tuning a language model in a situation that there is no labeled data is targeted in the second research question. Implementing an end-to-end semantic search solution in an MLOps framework corresponds to the third question. The fourth research question addresses the comparison between the performance of a large general-purpose language model with a very smaller one which is a fine-tunned version of the BERT. This model is fine-tuned on 'Semantic Text Similarity Benchmark' dataset to improve the accuracy of the nearest-neighbor search. Besides those stated goals, the capacity of the models in natural language understanding is assessed implicitly by ingraining the initial requirements in designing the test query set where the fifth research question is targeted.

Research questions 2 and 4 should be equipped with an annotated list of records determining if they are relevant to the query or not. As the scarcity of annotated datasets is a serious dilemma in real-world situations in the current research a dataset with the same structure is chosen to resemble the actual setting of designing and developing a solution based on intelligent information retrieval. Therefore, it is essential to propose a solution to deliver a judgment list for each query in the test query set. Section 3.3.3 covers this topic and describes three methods to resolve the lack of labels.

In compliance with creating a semantic search or in general an information retrieval system using LM embeddings, re-ranking is the followed-up phase in the architecture. As it is described in questions 1, 2, and 4, to achieve the determined goals, re-ranking would be beneficial. So, the Cross-Encoder described in Section 3.1.2.2 and the BERTScore explained in detail in Section 3.1.2.3 are utilized to re-rank the candidate list suggested by the Bi-Encoder. This process can improve user satisfaction in the cost of consuming more resources therefore it will be done on a smaller set of content that is the retrieved outputs by the Bi-Encoder. Figure 1 depicts the process that is followed and implemented in the current research as an attempt to respond to research questions.

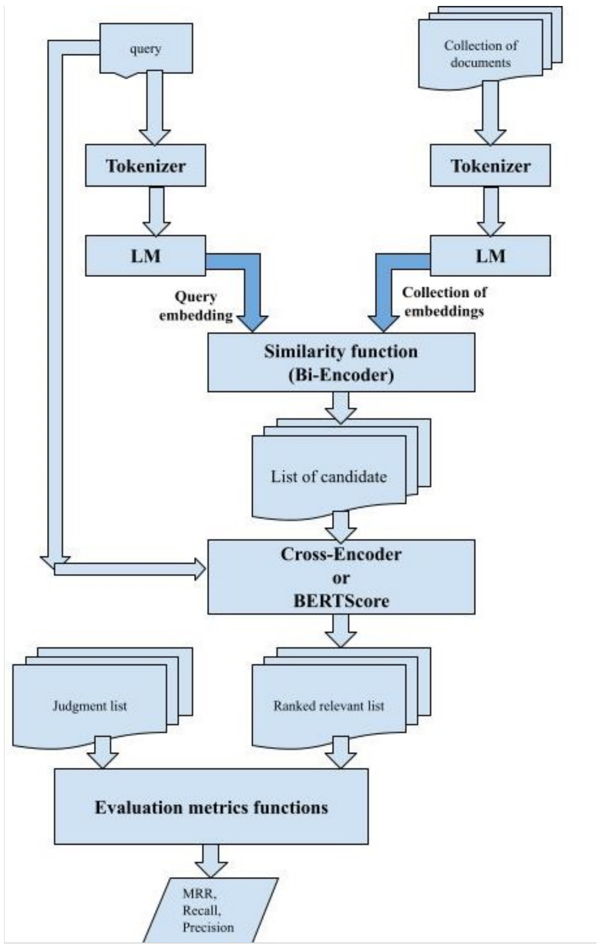


Figure 1. The research architecture

The judgment list shown in Figure 1 is produced by the introduced methods in Section 3.3.3. The LM represents the employed language model that can be Sentence-BERT or GPT-2 and the tokenizer is the matched one with the applied language model. The detail of each language model and its requirements are described in Section 2.3.5.

The model ability in natural language understanding will be analyzed corresponding to the outputs of both models and re-rankers which is targeted in question 5.

The major objective of applying a suitable language model to an information retrieval or specifically a semantic search pipeline is to develop an outstanding recommendation or search application that is efficient regarding resource consumption, latency, cost, and user satisfaction. Because of that, it is substantial to develop an automatic approach to evaluate the performance of the pipeline as a basis for comparison to select the most suitable model for the targeted task and available dataset. Such applications can be employed in various use cases so the procedure should be executed and monitored in a well-managed workflow all the way from data ingestion to deployment in production. MLOps is the solution to drive this situation which is the sought objective in research question 3.

"MLOps stands for Machine Learning Operations. MLOps is the core function of Machine Learning engineering, focused on streamlining the process of taking machine learning models to production and then maintaining and monitoring them" (*What Is MLOps?*, 2021).

In Figure 2 the cycle of MLOps illustrates the principal activities required to deliver a Machine Learning solution into production which includes Data Preparation, Development, Train or Re-train, Review, Deploy, Inference, Monitor, and Exploratory Data Analysis (EDA).

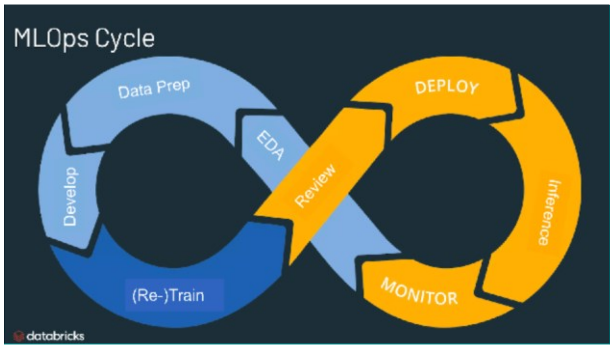


Figure 2. MLOps cycle

The thesis structure in nutshell is as follows: in Chapter 2 the significant tools and approaches utilized in Natural Language Process pipelines is discussed which includes tokenization as a preprocessing task before extracting the knowledge from contents in the form of word embeddings, Generative models which are the crucial type of machine learning algorithms in text generation and in the last section of this chapter the main functional elements of the Large Language Models in NLP pipeline are discussed which are categorized in Generative models as a sub-class of Deep Learning algorithms. Chapter 3 introduces the semantic search as it is studied in this research as an extensively applicable feature in many use cases. This chapter is divided into 3 sections to cover the primarily stages required to accomplish the semantic search workflow. This workflow initiates retrieval, ranking, and re-ranking to create results. Then, the performances measurement resides in the next section as the second phase in the flow to coherent the comparison between various models utilized in the retrieval. The last section discusses the initial asset in the any machine learning solution which is data therefore, selection or providing a proper dataset is the preliminary task. Solution implementation begins with loading the dataset, data exploratory analysis to get insight into the data, and cleaning and transforming it to make it as a proper dataset to be fed into the model. Data annotation requirement for evaluation and training besides the data labeling techniques are proposed in this section as well. The final subsection in this section covers another crucial aspect of input data, especially user query, which is addressed under the title of "Query Attributes".

In Chapter 4, main steps as a holistic workflow and implementation details are introduced in Sections 1 and 2, followed by presentation and discussion around outcome results in Section 4.3. Section 4.3 is divided into subsections based on the qualitative objectives determined in the research for example presenting the results of Bi-Encoder besides various applied re-rankers including Cross-Encoder and BERTScore to provide the baseline for comparison and analysis. The same logic is followed by tabulating the outcomes of the proposed labeling methods to arrange the reference for assessing their performance. Moreover, Sections 4.3.4 and 4.3.5 discuss and evaluate the performance of applied pipelines in this research with the related ones in the literature.

2. Tools and Initiatives of Significance

2.1 Parts of NLP Pipeline Utilized in Semantic Search

2.1.1 Tokenization

Although language models are unable to directly read and process raw text, most data in NLP tasks are presented in this form. To bridge this gap, tokenization is employed to convert the text into numerical format since language models operate solely with numbers. Various approaches exist to accomplish this task, each aiming to derive the most informative representation. This chapter explores three distinct algorithms including word-based, character-based, and sub-word-based with the objective of identifying the most meaningful encoding method.

2.1.1.1 Word-based Tokenization

This process involves splitting the text based on spaces or specific rules like punctuation marks. If the word-based tokenization is applied to the sentence "Let’s learn tokenization!" it will be split into "Let’s", "learn", and "tokenization" regarding space criterion and will be split into "Let", "’s", "learn", "tokenization", and "!" with respect to punctuation rule.

This approach is interesting, as the model has representations that are based on entire words. This algorithm assigns a unique identifier or "ID" to each word, allowing the model to have representations based on complete words. This approach offers advantages as a single number can convey significant contextual and semantic information contained within a word in a sentence. However, it has its limitations. For instance, similar words like "dog" and "dogs" are treated as entirely different due to distinct IDs, preventing the model from understanding their relatedness and the fact that "dogs" represents the plural form of "dog". Another challenge is the vast number of words in a language. If a model aims to learn all possible sentences and consider an ID for each different word, the number of words known as vocabulary size can quickly become huge. This a challenge because each ID is mapped into a large vector representing the word’s meaning and considering all these mappings with a huge vocabulary size demands a big number of weights.
To address this, we can make our tokenizer more efficient by excluding certain words that may not be essential. For example, during tokenizer training on a text, we can select the 10,000 most frequent words instead of including every word in the text or the entire language. This approach

creates a basic vocabulary where the tokenizer can convert these 10,000 words into numbers, while any other word is represented as an "out-of-vocabulary" or "unknown" word. However, this compromise means that all unknown words will have the exact same representation, resulting in information loss if there are numerous unknown words present (*Summary of the Tokenizers*, n.d.).

2.1.1.2 Character-based Tokenization

Instead of dividing the text into words, this approach involves segmenting it into individual characters. Languages generally have a vast number of distinct words, but the number of characters remains relatively small. For instance, the English language comprises approximately 170,000 different words, requiring a considerably large vocabulary to cover them all. In contrast, a character-based vocabulary can suffice with only 256 characters, including letters, numbers, and special characters. Even languages such as Chinese, which have many characters, typically have dictionaries containing up to 20,000 characters but over 375,000 different words.

Character-based vocabularies offer the advantage of requiring fewer unique tokens compared to word-based tokenization dictionaries. Additionally, these vocabularies tend to be more extensive than their word-based counterparts. By encompassing all characters used in a language, even unseen words during tokenizer training can still be tokenized, resulting in reduced occurrences of out-of-vocabulary tokens. Furthermore, this approach allows for the correct tokenization of misspelled words, instead of immediately categorizing them as unknown. However, it is important to note that this algorithm is not without limitations. One limitation is that characters do not convey as much information as a whole word does even though This is not generally applicable to all languages. While ideogram-based languages often convey substantial information within individual characters, languages based on the Roman alphabet, for example, require the model to comprehend multiple tokens to extract the meaning of a single word. Consequently, character-based tokenizers encounter another challenge. The resulting sequences are translated into a significant number of tokens, which can impact the model's context size and reduce the amount of text that can be utilized as input. Despite these considerations, character-based tokenization has demonstrated promising outcomes in the past and should be taken into account when tackling new problems, as it addresses certain issues encountered with word-based algorithms (*Summary of the Tokenizers*, n.d.).

2.1.1.3 Sub-word-based Tokenization

This approach falls between word-based and character-based tokenization methods, aiming to strike a balance between the drawbacks of both approaches. The objective is to mitigate the issues related to large vocabularies, a high number of out-of-vocabulary tokens, loss of meaning for similar words, and excessively long sequences.

These algorithms operate on the principle that frequently used words should remain intact as single units, while rare words can be broken down into meaningful sub-words. For example, the word "dog" would ideally be represented by a single token, rather than being split into individual characters. However, when encountering the word "dogs," the tokenizer should understand that it shares the same root as "dog" but with the addition of the letter "s," indicating a plural form while preserving the core concept.

Similarly, more complex words like "tokenization" can be decomposed into meaningful sub-words. The root of the word is "token," and "ization" serves as additional information that modifies the meaning. Consequently, it is sensible to split the word into "token" as the root (labeled as the "start" of the word) and "ization" as a complementary component (labeled as the "completion" of the word). This allows the model to comprehend the usage of "token" in various contexts and recognize the connections and similarities between words like "token," "tokens," "tokenizing," and "tokenization." Moreover, the model can infer those words sharing similar suffixes, such as "modernization" and "immunization," are likely to be used in similar syntactic situations.

Subword-based tokenizers typically employ a mechanism to distinguish between tokens that represent the beginning of words and tokens that complete words. Taking the example mentioned, the token "token" serves as the start of a word, while "##ization" completes the word. The prefix "##" signifies that "ization" is a part of a word rather than the initial component. The convention of using "##" as a prefix originated from the BERT tokenizer, which is based on the WordPiece algorithm. Other tokenizers may use different prefixes to indicate word parts or beginnings.

Numerous algorithms can be utilized for subword tokenization, and many state-of-the-art models in English leverage these approaches. They contribute to reducing vocabulary size by sharing information across different words and enabling the interpretation of prefixes and suffixes. These techniques ensure that meaning is preserved across similar words by recognizing the common tokens they consist of (*Summary of the Tokenizers*, n.d.).

In conclusion, sub-word-based tokenization has been widely used in NLP applications since its advantages include reasonable vocabulary size, the ability to learn meaningful context-independent representation, and proficiency in processing unseen words by decomposing them into known words.

There are various algorithms that have developed the idea of sub-word-based tokenization relying on training on the corpus that the corresponding model will be trained on. These algorithms consist of Byte-Pair Encoding (BPE), WordPiece, Unigram, and SentencePiece (*Summary of the Tokenizers*, n.d.).

2.1.1.3.1 Byte-Pair Encoding

Byte-Pair Encoding (BPE) is a popular sub-word tokenization technique used in natural language processing. It was introduced in 2015 by Sennrich et al. in a paper with the title Neural Machine Translation of Rare Words with Sub-word Units. It performs on the results of a pre-tokenizer that splits the training data into words. The pre-tokenizer can be as simple as space tokenization like what is utilized in GPT-2 and Roberta. More advanced pre-tokenizers use rule-based tokenization. Once the pre-tokenization stage is complete, a collection of distinct words is obtained, and their frequencies in the training data are recorded. Byte-Pair Encoding (BPE) proceeds by constructing a foundational vocabulary comprising all symbols present in the set of unique words. It then proceeds to learn merging rules, which combine two symbols from the base vocabulary to form a new symbol. This process continues iteratively until the desired vocabulary size is reached. It is worth noting that determining the desired vocabulary size is a hyperparameter that needs to be specified prior to training the tokenizer. It starts with a vocabulary containing individual characters or bytes and gradually merges the most frequently occurring adjacent pairs based on a predefined criterion, typically the likelihood of their co-occurrence. This merging process continues until a desired vocabulary size, or a predetermined number of merge operations is reached.

During tokenization, BPE splits words into sub-word units based on the learned merge operations. By breaking down words into sub-word units, BPE captures both the individual morphemes and the overall word structure, enabling models to handle complex words and unseen combinations effectively.

In this method, the vocabulary size i.e., the base vocabulary size and the number of merges is a hyperparameter to choose. For instance, GPT has a vocabulary of size 40478 since it has 478 base characters and training will be stopped after 40000 merges. GPT-2 uses bytes as the base vocabulary of size 256 ensuring that every base character is included with additional rules to handle the punctuations therefore, it has a vocabulary size of 50257 which corresponds to the 256 bytes base tokens, a special end-of-text token, and learning stopped after 50000 merges (Sennrich et al., 2016) and (*Summary of the Tokenizers*, n.d.).

2.1.1.3.2 WordPiece

The WordPiece is another form of sub-word tokenization algorithm introduced by Schuster et al. in 2012 and is similar to BPE. This algorithm first creates a vocabulary that contains every character in the training data and then starts learning a predefined number of merge rules. The difference between BPE and WordPiece is that in WordPiece symbol pairs are not selected based on their frequency but they are chosen based on maximizing the likelihood of the training data if they are present (*Summary of the Tokenizers*, n.d.).

2.1.1.3.3 Unigram

The unigram algorithm was introduced in 2018 by Kudo which considers a huge number of symbols in the initial vocabulary and progressively decreases the number of symbols in the vocabulary to reach the desired vocabulary size. Regarding that, this method performs entirely differently from BPE and WordPiece. The base vocabulary could consist of all pre-tokenizer words and the most common substrings. It should be noted that Unigram is not used directly but in collaboration with SentencePiece.

The Unigram training happens based on minimizing the loss function over training data for the given current vocabulary and unigram language model. The algorithm computes the loss function increment by removing the symbol from the vocabulary. This calculation will be repeated for each symbol in the vocabulary and then a certain percent of symbols that make the lowest loss increment would be removed from the vocabulary iteratively when the desired vocabulary size is the stop criterion.

The probability of each token in the training corpus will be saved on top of the vocabulary. accordingly, the algorithm can choose the most likely tokenization also it can offer possible tokenization according to their probability (*Summary of the Tokenizers*, n.d.).

2.1.1.3.4 SentencePiece

There is a common issue with all tokenization methods discussed so far. The problem is that they all consider space as the separator between words. Even though, it does not apply in all languages for example languages like Chinese, Japanese, and Thai need specific pre-tokenizer. SentencePiece introduced in 2018 by Kudo et al. has been a solution to this problem. In this method, the space would be included in the vocabulary set and the input to the algorithm is considered as raw input stream. Therefore, the BPE or Unigram will be utilized in collaboration with SentencePiece considerations to provide an appropriate vocabulary. ALBERT and T5 are two samples of the language models that use SentencePiece in combination with Unigram (*Summary of the Tokenizers*, n.d.).

2.1.2 Word Embedding

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on enabling computers to understand, interpret, and generate human language. It involves developing algorithms and models that can process and analyze textual data, enabling tasks such as text classification, sentiment analysis, machine translation, and question-answering.

NLP task is a challenging problem in Deep Learning because of a crucial requirement of a proper representation of words since the words are not understandable for computers. Therefore, some techniques should be utilized to characterize the words in numerical format.

Word embedding is a technique commonly used in NLP to represent words as dense and continuous vector representations in a lower-dimensional space. It aims to capture the semantic and syntactic relationships between words, allowing algorithms to interpret and reason about words based on their vector representations.

Traditionally, words in NLP were represented using one-hot encoding, where each word is represented as a sparse vector with a value of 1 for its corresponding index and 0 for all other indices. However, one-hot encoding lacks the ability to capture semantic relationships between words and suffers from high dimensionality.

To mitigate the drawbacks of the traditional techniques for word embedding, more recent methods learn word representations by training on large corpora of text. These models generate dense vector representations, where words with similar meanings are placed closer together in the embedding space. For example, words like "cat" and "dog" might have similar vector representations, reflecting their semantic similarity. This concept is depicted in Figure 3 where each word is mapped to a dense vector corresponding to its relationship to the surrounding words in the sequence and the concept in total. The dimensionality reduction in the figure is applied to show the closeness of vectors in a 2-D space.

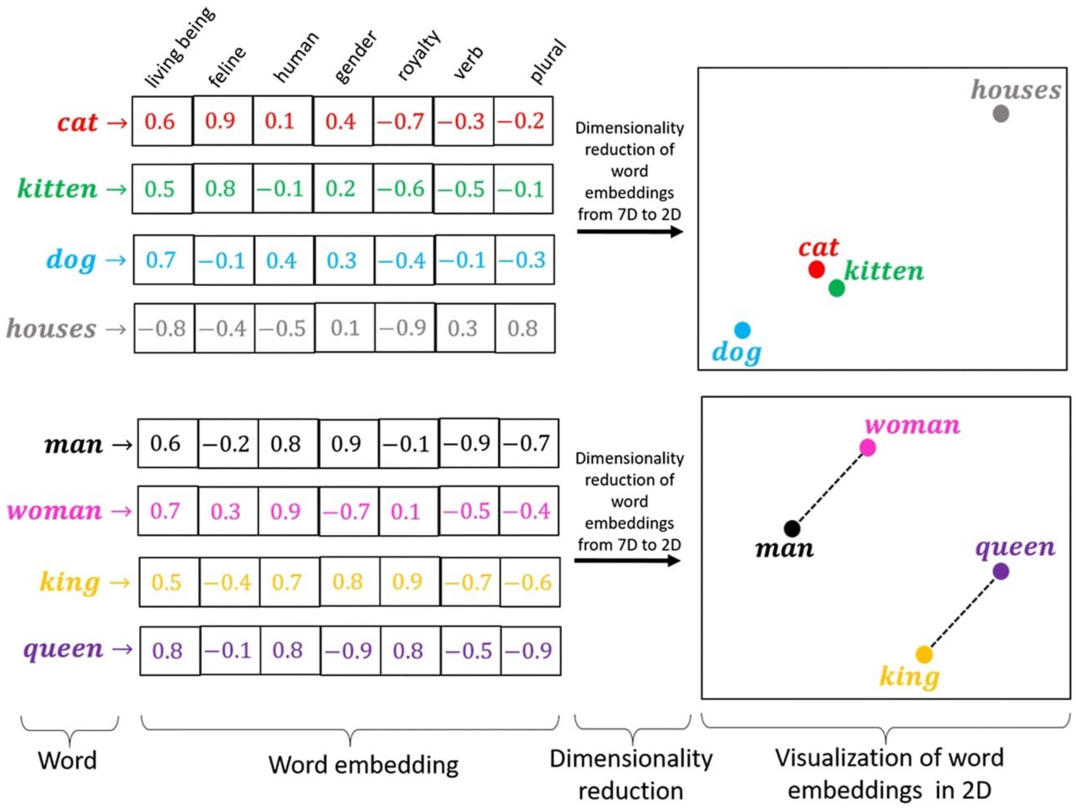

living being
feline
human
gender
royalty
verb
plural
cat → 0.6 0.9 0.1 0.4 −0.7 −0.3 −0.2
kitten → 0.5 0.8 −0.1 0.2 −0.6 −0.5 −0.1
dog → 0.7 −0.1 0.4 0.3 −0.4 −0.1 −0.3
houses → −0.8 −0.4 −0.5 0.1 −0.9 0.3 0.8
man → 0.6 −0.2 0.8 0.9 −0.1 −0.9 −0.7
woman → 0.7 0.3 0.9 −0.7 0.1 −0.5 −0.4
king → 0.5 −0.4 0.7 0.8 0.9 −0.7 −0.6
queen → 0.8 −0.1 0.8 −0.9 0.8 −0.5 −0.9
Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

Figure 3. Word Embedding (Solanki, 2022)

Pennington et al. introduced GloVe (Global Vectors for Word Representation) in 2014 as a vector space representation for words which can capture semantic and syntactic regularities. Glove is designated to capture the semantic relationships between words based on their co-occurrence in a large corpus of text. It constructs a word-context matrix based on word co-occurrence statistics. The matrix records how often words appear together in the same context within the corpus. The algorithm in this approach aims to learn word vectors that minimize an objective function that quantifies the difference between the dot product of word vectors and the logarithms of the observed word co-occurrence probabilities. When the algorithm is trained, it can produce word vectors of fixed dimensionality for each word in the vocabulary (Pennington, et al. 2014).

FastText is another technique for mapping words into embedding based on the Enriching Word Vector with Subword Information approach which is proposed by Bojanowski et al. in 2017. The idea of this approach is to overcome the limitation of the word representation models that ignore the morphology of the words.Specially it happens when a word representation algorithm is trained on large unlabled corpora of a language with large vocabularies and many rare words. Therefore, the proposed method is formed based on skipgram model, where each word is represented as a bag of characters n-grams, and a vector representation is associated to each character n-gram. In other words, it operates at the subword level. It first segment words into smaller units called character n-gram. FastText builds a vocabulary not only for whole words but also for these subword tokens. Then it learns only the vectors for each word by summing up the vectors of its constituent subword tokens. This composition captures the semantic meaning of the word based on the meanings of its constituent parts. The key advantages of this model are being fast and ability of compute embedding for words that did not appear in the training data (Bojanowski et al. 2017).

Word embeddings have several advantages in NLP tasks. First, they capture semantic relationships between words, allowing models to generalize better and make more accurate predictions. For example, with word embeddings, a model can infer the similarity between two words or even find analogies like "king" is to "queen" as "man" is to "woman". Second, word embeddings reduce the dimensionality of the representation space, making computations more efficient and less memory intensive. Lastly, they enable the transfer of knowledge from pre-trained word embeddings to downstream tasks with limited training data, improving performance even in data-scarce scenarios. Semantics of words is more beyond the semantic meaning of their constituent parts or subwords. There are many contextual complex relationships between words in sentences or even paragraphs and wider text structures that needs to be captured. The more contextual information encoded in word representations, the more successful NLP downstream task can be delivered. Large Language Models (LLM) are used to encode words and other terms into vectors based on their context in sentences or larger text according to their capacity for learning.

2.2 Generative Models Utilized in NLP Tasks

Generative artificial intelligence (AI) describes algorithms, such as ChatGPT, that can be used to create new content, including audio, code, images, text, simulations, and videos.

Generative models are a type of unsupervised machine learning model that aims to create or generate new data that is like a given training dataset. These models learn the underlying patterns, structure, dependencies, and relationship of the training data and then use that knowledge to generate new samples that resemble the original data. A Generative AI model starts by efficiently encoding a representation of what you want to generate. For example, a Generative AI model for text might begin by finding a way to represent the words as vectors that characterize the similarity between words often used in the same sentence.

In other words, Generative Models approximate the distribution of the data, by observing the training data through the probabilistic lens, which summarizes the information about the dataset within a finite set of parameters.

To train the Generative Model, there must be a huge amount of data because the significant consideration in training this type of model is a dramatically larger number of training data than the number of model parameters. Therefore, the models are forced to discover and capture the features and distribution of the training data automatically in order to generate it (create samples from the model). Regarding the concept of training Generative Models, it should be noticed that there is not a simple supervised setting and explicit desired labels. The goal is to come adequately close to the real samples.

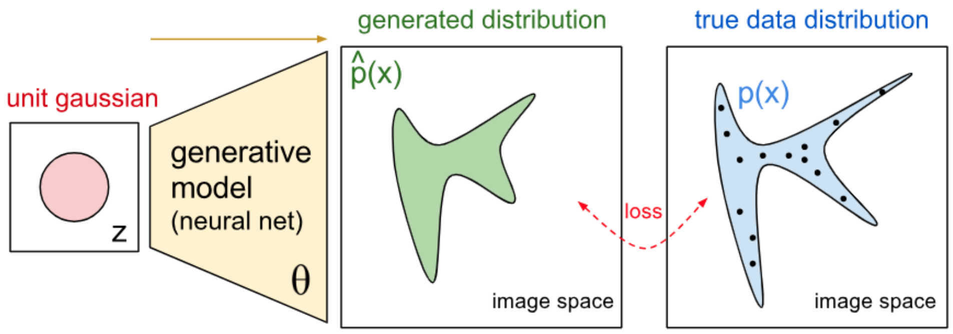


Figure 4. Generative models mathematical definition (OpenAI GPT-3, 2022)

To describe Generative Models mathematically, a dataset of examples would be defined as $x_1, x_2, \ldots, x_n$ samples of a true data distribution $p(x)$. The model also describes a distribution $p_{\theta}^{\wedge}(x)$ that is defined by drawing samples from unit Gaussian distribution and mapping them through a neural network. The parameter θ determines the modification of the generated distribution. The final goal is to find the parameter θ that minimizes the distance between the generated distribution and the true one. The mathematical definition is illustrated in Figure 4 (*OpenAI GPT-3*, 2022).

There are three approaches in generative models including Generative Adversarial Networks (GAN), Variational AutoEncoder (VAE), and Autoregressive.

2.2.1 Generative Adversarial Network (GAN)

As a variation of Generative models, A Generative Adversarial Network (GAN) is a type of AI model that consists of two neural networks to estimate the generative model in an adversarial process: a generator (G) which is a generative model that captures data distribution and a discriminator (D) that estimates the probability of a sample drawn from training data rather than G. The primary purpose of a GAN is to generate new data that is similar to a given training dataset. The generator network in a GAN learns to generate synthetic data by mapping random noise (unit Gaussian distribution) or input data to the desired output data. For example, in the case of generating realistic images, the generator network takes random noise as input and learns to generate images that resemble the training dataset. Initially, the generated data may not resemble the real data, but as the model trains, it improves its ability to produce more realistic outputs.

The discriminator network, on the other hand, acts as a critic and tries to distinguish between real and generated data. It learns to classify whether a given input is real or fake by receiving both real data from the training set and generated data from the generator network. The discriminator provides feedback to the generator, helping it improve its generated outputs over time.

During training, the generator and discriminator networks engage in a competitive process. The generator aims to produce synthetic data that is indistinguishable from real data, while the discriminator aims to accurately classify between real and generated data. Through this adversarial training process, both networks gradually improve their abilities, leading to the generation of high-quality data samples. The adversarial procedure in training a GAN is illustrated in Figure 5.

One of the significant advantages of GANs is their ability to capture complex patterns and generate data that closely resembles the training dataset. GANs have been successfully used for tasks such as image synthesis, video generation, text generation, and even generating new artwork. However, training GANs can be challenging, as they require careful balancing and tuning of the generator and discriminator networks to achieve stable and desirable results (Goodfellow et al., 2014).

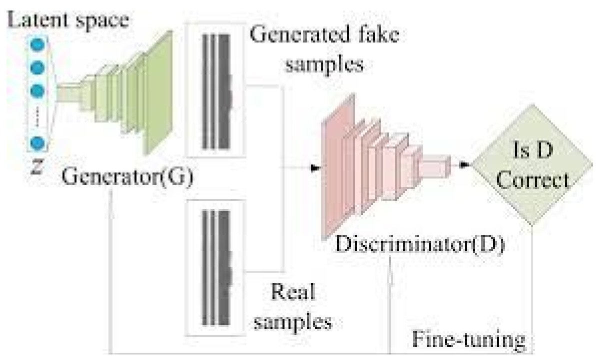


Figure 5. The adversarial procedure in training a GAN (What's GAN, n.d.)

2.2.2 Variational AutoEncoder (VAE)

Variational AutoEncoder is a type of generative model that combines elements of both autoencoders (a type of unsupervised learning approach) and probabilistic modeling. Autoencoders are not used to generate data but, they are an approach for learning a lower dimensional feature representation from unlabeled training data. So, if there is input data X and feature Z is going to be learned, there will be an encoder to map the input data to feature Z. The encoder can take various forms even though it would be usually implemented using neural networks. As it is mentioned, the dimensionality of Z would be smaller than X which means that the encoder should just map the most meaningful factors of variation in the data into Z (Stanford University School of Engineering, 2017).

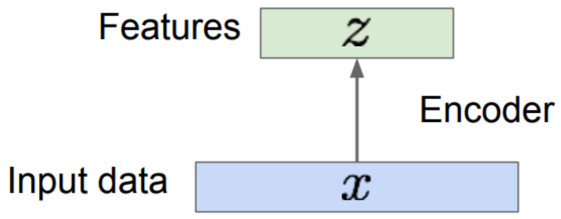


Figure 6. Autoencoder structure (Li et al., 2017)

The model is trained so that the features can be used to reconstruct the original data with another network called Decoder. The Decoder network's output ($\hat{X}$) would be the same size as the input data. The Autoencoders are usually symmetric which means both Encoder and Decoder are implemented with the same neural network architecture. Figure 4 demonstrates the process of the reproduction of input data from learned feature representation. In the training procedure, the squared error loss function (L2 Loss function) will be minimized in order to reconstruct the input data ($\hat{X}$) from feature representatives (Z). The loss function is in the form of $||x - \hat{x}||^2$. It should be noted that the feature representation Z could be used for supervised learning. In other words, the Decoder layer could be replaced by a classifier which is shown in Figure 8 (Stanford University School of Engineering, 2017).

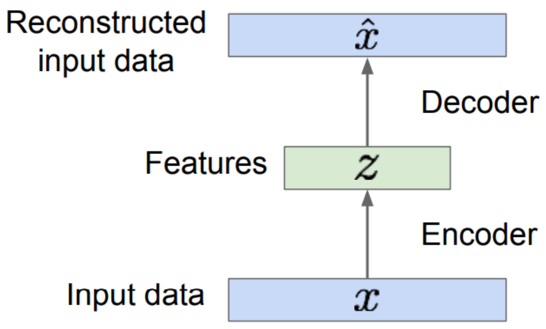


Figure 7. Reproduction of input data (Li et al., 2017)

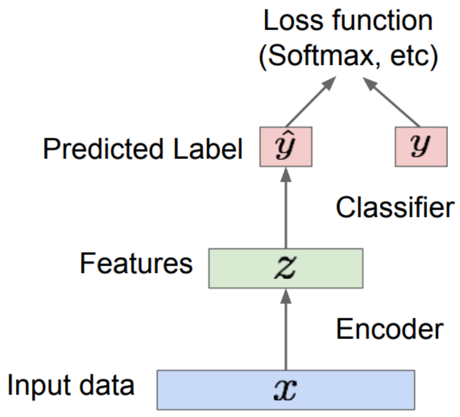


Figure 8. Autoencoder feature representation for classification (Li et al., 2017)

The feature space Z could be considered as a latent variable that captures the factors of variation in the training data. The goal is to generate new data samples from that. Therefore, to generate X from an unobserved latent space Z, it is required to make an assumption for the latent space distribution which could be Gaussian distribution to draw samples from Z formulated as $p_{\theta^*}(z)$, and then X would be sampled utilizing the conditional distribution defined as $p_{\theta^*}(x|z^{(i)})$ which would be implemented as a Decoder neural network. According to the applying distributions to sample from the model in order to generate data, Variational Autoencoder is considered as a probabilistic spin on Autoencoders that is formulated as follows.

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) d_z$$

The model parameters should be learned to maximize the likelihood of the training data. The problem in this formulation is that the integral is intractable, and the likelihood cannot be computed. Besides that, the posterior density $p_\theta(z|x) = p_\theta(x|z)p_\theta(z) / p_\theta(x)$ is also intractable. The proposed solution is to define an Encoder network that approximates $p_\theta(z|x)$ that derives a lower bound on the data likelihood which can be optimized. The encoder network is also called "recognition or inference" and the decoder is called "generation".

The samples generated using Variational autoencoders are blurrier and of lower quality compared to GANs (Kingma & Welling, 2022).

The encoder network encodes the input data into two vectors: the mean (μ) and the standard deviation (σ) of the latent distribution. These vectors represent the parameters of a multivariate Gaussian distribution in the latent space.

Next, a random sampling step is introduced to sample a point (z) from the Gaussian distribution defined by μ and σ. This sampling is done to make the model stochastic, allowing it to generate diverse outputs for a given input.

The sampled point (z) from the latent space is passed through a decoder network to generate the reconstructed data.

A crucial aspect of VAEs is the choice of loss function. VAEs use a loss function that encourages the latent space to follow a standard Gaussian distribution and ensures that the reconstructed data closely matches the input data.

The divergence between the distribution of the latent space (defined by μ and σ) and a standard Gaussian distribution is measured with KL Divergence Loss. It encourages the latent space to be well-behaved and encourages the network to learn a meaningful representation.

During training, VAEs aim to minimize the combined loss (reconstruction loss and KL divergence loss) by adjusting the parameters of both the encoder and decoder networks.

After training, the VAE can be used for data generation by sampling points from the latent space and passing them through the decoder network to produce new data points. By interpolating between latent space points or exploring nearby regions, the VAE can generate variations of data like the training data. VAEs are often applied in tasks like data generation, image denoising, and dimensionality reduction.

2.2.3 Autoregressive

Autoregressive models are a class of statistical models used to analyze and forecast time series data. The fundamental idea behind autoregressive models is that the value of a variable at a given time point can be predicted based on its previous values. It could be said that it is a regression of a variable against itself over time. The autoregression models of order p can be composed as follows where p determines how many past values are included in the model.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

An autoregressive generative model is a type of generative model that learns to generate new data by modeling the conditional probability distribution of each data point given the previous data points. In other words, it generates data by considering the dependencies and relationships between the elements in a sequence. From this point of view, they can be compared with sequential models like Recurrent Neural Networks (RNN).

Autoregressive models are commonly used in fields such as natural language processing and time series analysis, where the order and context of the data play a crucial role. These models can capture complex patterns and generate new samples that exhibit similar characteristics to the training data. The key idea behind autoregressive models is that each element in the sequence is generated based on a learned probability distribution conditioned on the previous elements $p(x_{t+1}|x_1, x_2, \ldots, x_t)$.

A typical example of such models is GPT. GPT is the first autoregressive model based on the transformer architecture, pre-trained on the Book Corpus dataset.

2.2.4 Large Language Models

2.2.4.1 Foundational Models

Foundation Model is described as follows by Wikipedia:

A Foundation Model (also called Base Model) is a large Machine Learning (ML) model trained on a vast quantity of data at scale (often by self-supervised learning or semi-supervised learning) such that it can be adapted to a wide range of downstream tasks. Foundation Models have helped bring about a major transformation in how AI systems are built, such as by powering prominent chatbots and other user-facing AI. The Stanford Institute for Human-Centered Artificial

Intelligence's (HAI) Center for Research on Foundation Models (CRFM) popularized the term (“Foundation Models,” 2023).

A foundation model, in the context of large language models (LLMs), refers to a base model that serves as the starting point for developing more specialized or domain-specific models. It is designed to capture general knowledge and language understanding across a wide range of topics and tasks. A foundation model is typically pre-trained on vast amounts of text data to learn patterns, context, and relationships within the language.

Foundation models act as the building blocks for various downstream applications and tasks. They provide a strong foundation of language understanding that can be fine-tuned and adapted to specific use cases, such as translation, sentiment analysis, question answering, and more. By utilizing transfer learning, where the knowledge gained during pre-training is transferred to specific tasks, foundation models enable efficient development and deployment of language-based AI systems.

Large language models (LLMs) are a type of foundation model that has been scaled up in size and capacity, often with millions or even billions of parameters. LLMs, such as GPT-3, form enhanced language capabilities on a wide range of natural language processing tasks. They have become the focus of intensive research and development efforts, driving advancements in the field of AI and revolutionizing the way language-based algorithms are developed and utilized.

2.2.4.2 Language Models (LM)

A language model is an artificial intelligence (AI) system designed to understand and generate language-based content. It uses statistical and probabilistic techniques to predict the next word or sequence of words in each context in the form of a sequence of words occurring in a sentence. By capturing the patterns, structures, and semantics of the language. Language models are crucial in enhancing computers’ ability to comprehend and generate human-like content.

Text generation is one of the core functions of language models. It involves generating coherent and contextually relevant text on a given input. Language models achieve this by learning from large amounts of text data and acquiring the ability to predict the most probable next word or phrase. Through this process, they can generate meaningful sentences, paragraphs, and even entire articles. In simple words, a language model predicts the next word in a sequence.

Language Models have many applications for instance: Part of Speech (PoS) tagging, Machine Translation, Text Classification, Speech Recognition, Information Retrieval, New Article Generation, Question Answering, etc.

2.2.4.3 Large Language Model (LLM)

Large language models and generative models are closely related in the field of artificial intelligence and machine learning, with large language models often serving as a specific type of generative model specialized in generating human-like text. They have played a significant role in advancing natural language processing tasks and have demonstrated their generative capabilities across various applications. While generative models encompass a broader category, large language models are a prominent subset, particularly renowned for their text generation capabilities.

They are trained on an extensive amount of data and has enormous parameters. These models can generate highly coherent and contextually relevant text due to their immense size and training data. Large language models, such as OpenAI's GPT-3, have billions of parameters and are trained on massive text corpora, enabling them to exhibit remarkable language understanding and generation capabilities.

While a language model refers to the general concept of a model that understands and generates human language, a large language model specifically highlights the scale and complexity of the model due to its extensive training and parameter size.

The training process for a large language model involves exposing it to diverse and extensive text corpora, such as books, articles, and internet data. By analyzing this vast amount of text, the model learns to predict the likelihood of certain words or phrases following a received input.

Large language models have demonstrated impressive language understanding and generation capabilities. They can generate realistic and contextually appropriate responses in various applications, including chatbots, virtual assistants, content creation, Question Answering systems, and language translation. Due to their extensive training and parameter size, large language models have the potential to revolutionize how we interact with AI systems and consume information.

2.2.4.4 How Does a Language Model Work

Language Models as an approach to developing AI algorithms to comprehend and generate language have been widely studied in the past two decades. The approach evolved from statistical

models to deep neural models. These models are designed to predict the next word in the sentence by learning the features and characteristics of the language by examining the texts feed to it.
An LM assigns probabilities to the sequence of arbitrary symbols in a way that maximizes the probability of that sequence $(w_1, w_2, \dots, w_n)$ in a specific language which would be formulated as follows (*OpenAI GPT-3*, 2022):

$$P(w_1, w_2, \dots, w_n) = p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1, w_2) \dots p(w_n \mid w_1, w_2, \dots, w_{n-1}) = \prod_{i=1}^{n} p(w_i \mid w_1, \dots, w_{i-1})$$

The sentence "Where are we going" can be considered as a sequence of words that the last word, "going", is the target to be generated as the next symbol in the sequence.

$$S = Where\ are\ we\ going$$
$$P(S) = p(Where) \times p(are \mid Where) \times p(we \mid Where\ are) \times p(going \mid Where\ are\ we)$$

The probability of a sentence can be defined by the product of the probability of each symbol in the condition of the previous sequence of symbols.

2.3 Deep Learning Approaches Utilized in NLP Pipeline

Deep Learning is a subfield of machine learning that has revolutionized Natural Language Processing in several ways by enabling the development of models that can understand, generate, and process human language more effectively, leading to advancements in various NLP applications. These models have transformed the way we interact with and extract insights from textual data. One Deep Learning approaches that has been widely utilized in NLP tasks is Word Embedding techniques like GloVe and FastText discussed in the Section 2.1.2. Other approaches including Recurrent Neural Networks that can produce a sequence of data, Transformer Models and Attention mechanism to understand the context and relationship between words in a sentence, and Transfer Learning are introduced in the following sections in this chapter.

2.3.1 RNN and Sequence-to-Sequence Models

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes can create a cycle, allowing the output from some nodes to affect subsequent input to the same node. This allows it to exhibit temporal dynamic behavior (“Recurrent Neural Network,” 2023).

RNNs process sequential data by retaining and utilizing information from previous steps. Unlike feedforward neural networks that process each input independently, RNNs have internal memory that allows them to consider the context and dependencies among the elements in a sequence besides they can handle variable-length sequences, maintain information about the order and share parameters across the sequence.

The fundamental building block of an RNN is the recurrent neuron, which takes an input along with its own previous output and uses them to compute the current output. This recurrent structure enables RNNs to maintain a hidden state that captures the information from previous inputs and influences the computation of subsequent outputs. A single RNN cell is illustrated in Figure 9.

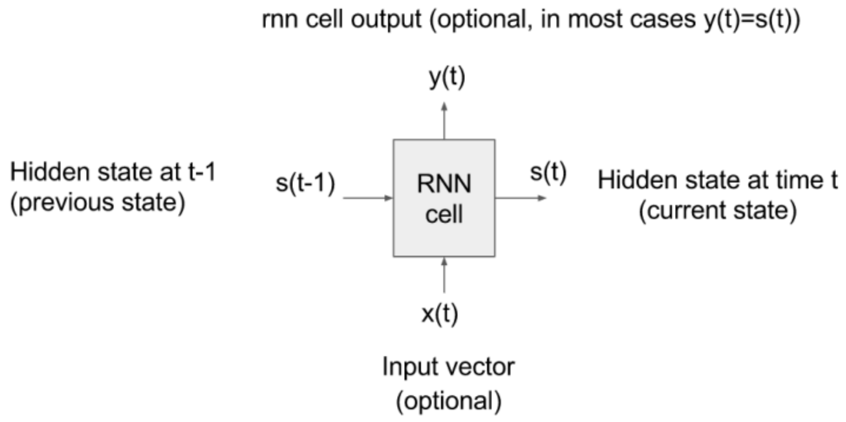

rnn cell output (optional, in most cases y(t)=s(t))
y(t)
Hidden state at t-1 (previous state)
s(t-1)
RNN cell
s(t)
Hidden state at time t (current state)
x(t)
Input vector (optional)

Figure 9. RNN cell (Khriyenko, 2023)

RNNs are well-suited for tasks that involve sequential data, such as natural language processing, speech recognition, time series analysis, and handwriting recognition. They perform well in modeling and generating sequences, as they can capture long-term dependencies and learn patterns over time.

RNNs come in many variants including Vanilla, Gated Recurrent Unit (GRU), and Long Short-Term Memory (LSTM).

The basic RNN is a fully connected RNN where the outputs of all neurons connect to the input of all neurons. Fully connected RNN (FRNN) is shown in Figure 10. In other RNN topologies, the weights of some connections are set to zero to lead to a disconnection between those neurons.

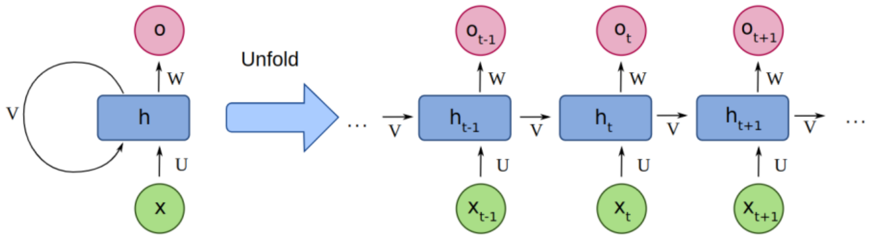


Figure 10. Fully connected RNN (“Recurrent Neural Network,” 2023)

One important variant of RNNs is the Long Short-Term Memory (LSTM) network. LSTMs address the vanishing gradient problem that can occur in traditional RNNs by introducing specialized memory cells that selectively retain or forget information over time which are called “forget gates”. LSTMs have become popular in many applications due to their ability to effectively capture long-term dependencies and mitigate the vanishing gradient issue. Figure 11 demonstrates the architecture of LSTM. The LSTM is significantly capable of deciding on how much of each component should be let through by the gates structures which are shown in Figure 11 by F_t, I_t, and Q_t for Forget gate, Input gate, and Output gate respectively. Forget gate decides what information should be thrown away, the Input gate determines what new information should be added to the cell, and the Output gate indicates what would be the output.

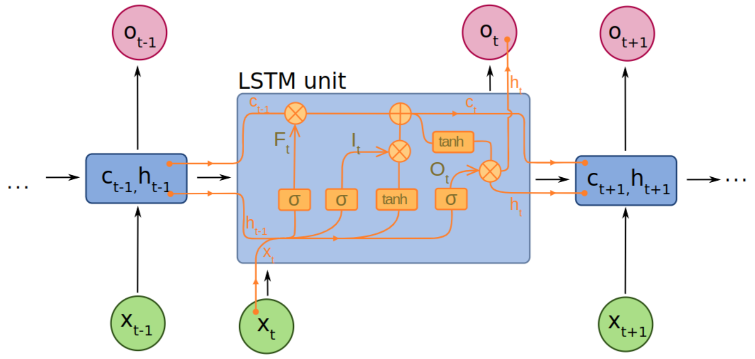


Figure 11. LSTM architecture (“Recurrent Neural Network,” 2023)

Another variant is the Gated Recurrent Unit (GRU), which is similar to LSTM but with a simplified architecture. It combines the forget and input gates into a single gate called the Update gate and merges the cell and hidden state. GRUs also address the vanishing gradient problem and have shown comparable performance to LSTMs in various tasks while being computationally efficient. GRU architecture illustrated in Figure 12 has fewer parameters than LSTM and does not include any output gate which means the output cannot be filtered.

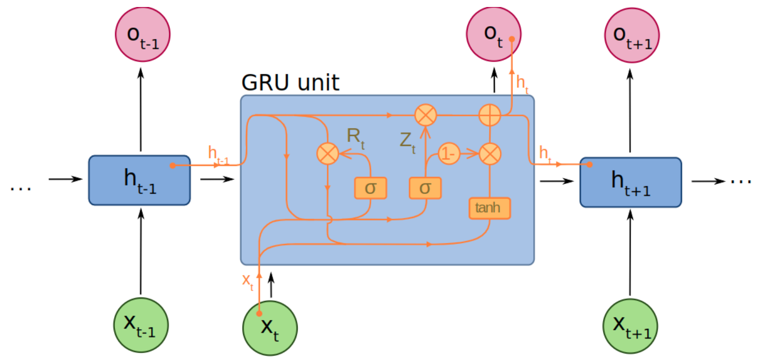


Figure 12. GRU architecture ("Recurrent Neural Network," 2023)

There is another architecture of RNNs that can traverse in the reverse direction also. Bi-directional RNNs consider not only the context of the previous occurrences but also the future context in prediction by connecting two hidden layers to receive the information from backward and forward simultaneously to the same output. The idea is derived from the fact that the output at time *t* depends on both past and future elements that appear in the sequence. The architecture of BiRNN would be described as two RNNs stacked on top of each other in different directions leading to the collaboration of both hidden states in computing the output. Figure 13 demonstrates the Bi-directional RNN with two hidden layers of opposite directions to the same output.

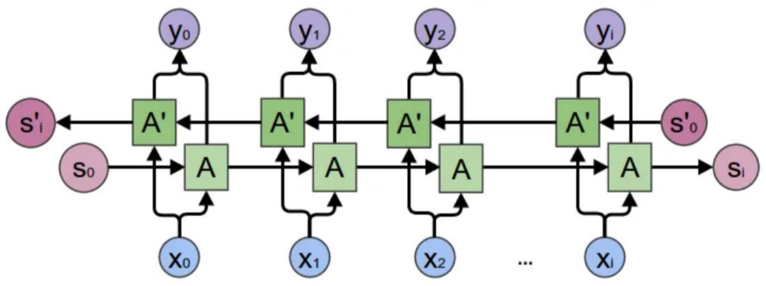


Figure 13. Bi-directional RNN (Olah, 2015)

Training RNNs involves updating the network's parameters using gradient-based optimization methods, such as backpropagation through time (BPTT), which is an extension of the standard backpropagation algorithm. BPTT computes gradients by unfolding the recurrent structure of the network over time and propagating errors back through each time step.

RNNs can be used for both supervised and unsupervised learning tasks. In supervised learning, RNNs can be trained with labeled data to perform tasks like sequence classification, named entity recognition, and machine translation. In unsupervised learning, RNNs can be used for tasks like language modeling, where the network learns to predict the next word given previous words.

While RNNs have been successful in many applications, they also face challenges. They can be computationally expensive to train and prone to issues like vanishing or exploding gradients, which can affect learning and model performance. Additionally, RNNs may struggle to capture very long-term dependencies due to the limitations of their recurrent structure.

2.3.1.1 Sequence-to-Sequence Models

Sequence-to-Sequence (Seq-to-Seq) models are an extension of RNNs designed specifically for tasks involving variable-length input and output sequences. Deep learning models like the Seq2Seq model, with the help of attention mechanisms, are used for tasks like machine translation and text summarization. Figure 14 depicts some examples of the use cases.

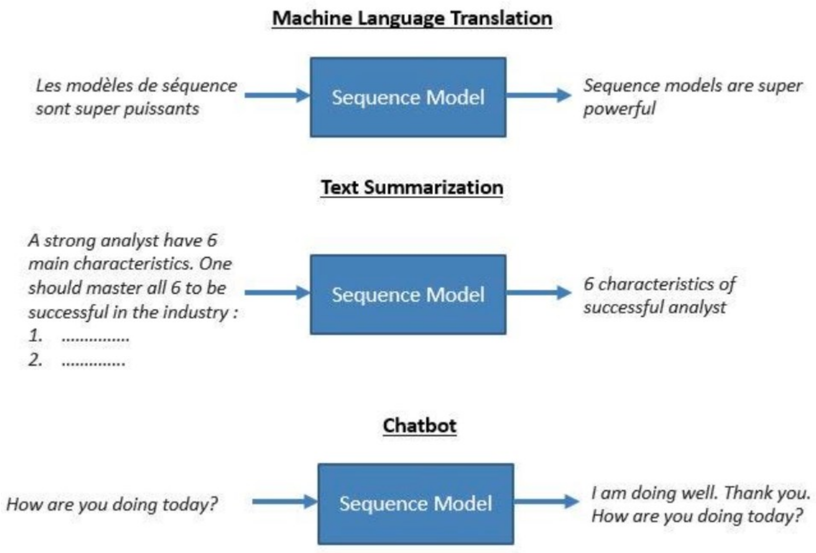

Machine Language Translation
Les modèles de séquence sont super puissants
Sequence Model
Sequence models are super powerful
Text Summarization
A strong analyst have 6 main characteristics. One should master all 6 to be successful in the industry :
1. ....
2. ....
Sequence Model
6 characteristics of successful analyst
Chatbot
How are you doing today?
Sequence Model
I am doing well. Thank you. How are you doing today?

Figure 14. Seq2Seq model use cases (guest_blog, 2020)

Despise the Deep Neural Networks (DNN) are extremely powerful in machine learning to capture input features and create outstanding performance, they can only be applied to inputs with fixed lengths. For tasks like question-answering where each of the questions and answers is formed in sequences of various lengths and the length of the sequence is not known preceding, the DNNs are not applicable. In such cases, a domain-agnostic method that could be trained to map sequences to sequences would be effective.

The main concept of implementing Seq2Seq models is to employ two recurrent neural networks (RNNs), an encoder RNN and a decoder RNN. The encoder processes the input sequence step by step, generating a fixed-length representation of context. This context vector contains the encoded information from the entire input sequence and serves as the starting point for the decoder.

The decoder RNN takes the context vector as its initial hidden state and generates the output sequence step by step. At each time step, the decoder takes the previously generated token (word, character, or other unit) as input and produces the next token in the output sequence. This process continues until an end-of-sequence token is generated or a maximum length is reached. The Seq2Seq model architecture is illustrated in Figure 15.

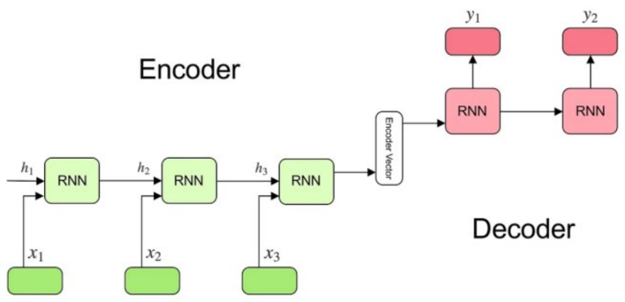


Figure 15. sequence to sequence model (Nadeem, 2021)

Seq2Seq models can be implemented using different types of RNNs, such as Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRUs), as encoder and decoder units. These RNN units allow the model to capture and process sequential information effectively. The LSTM can take sentences of different lengths as input and convert them into fixed-dimensional vector representations. For example, in machine translation, since translations are often rephrased of the source sentences, the LSTM is seeking to find sentence representations that effectively capture their meaning. This is because sentences with similar meanings are positioned closer to each other in the vector space, while sentences with different meanings are placed further apart. The similarity between the sequences could be computed and compared with respect to the fixed-length vector representation.

As has been discussed in the previous chapter, a standard RNN computes a sequence of outputs $(y_1, \ldots, y_T)$ regarding a given sequence of inputs $(x_1, \ldots, x_T)$ by iterating over the following equation:

$$h_t = sigm(W^{hx}x_t + W^{hh}h_{t-1})$$
$$y_t = W^{yh}h_t$$

The problem is that when the input and output have different lengths and complex relationship, it will not be straightforward how to apply the RNN. The solution to this problem is to map the input sequences to a fixed-length representation which is called v in the formulation. Then map that representation to the output sequence utilizing another RNN (decoder) which computes the conditional probability of the output (Sutskever et al., 2014).

$$p(y_1, \ldots, y_{T'} | (x_1, \ldots, x_T)) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

Even though Seq2Seq models have shown impressive performance in various sequence generation tasks, they may face challenges when dealing with very long sequences and dependencies, happens to gradient vanishing or gradient explosion. In addition, they need long training steps because the optimization regarding many parameters takes a long time, and the size of the network depends on the length of the sequence. The last but not the least issue with Seq2Seq models is that RNNs are not suitable for parallel computation. All the computations in the RNNs network occur sequentially and cannot be parallelized.

One important extension to Seq2Seq models is the attention mechanism. The attention mechanism allows the decoder to focus on specific parts of the input sequence when generating each token in the output sequence. This mechanism helps the model handle long sequences and improves the performance besides fitting parallelization (Sarkar, 2022).

2.3.2 Attention

The attention mechanism in transformers is a fundamental component that enables the model to focus on relevant parts of the input sequence when making predictions. Unlike traditional sequence-to-sequence models that use fixed-length context vectors, transformers allow for more flexible and adaptive information processing.

According to what is explained in this chapter, RNNs as the foundation of Sequence-to-Sequence models, would encode the input sequence of the occurrences till time *t* to a hidden vector to be passed to the decoder to generate the output sequence at time *t+1*.

On the other hand, the Attention mechanism would not map the whole input sequence into a fixed-length vector. Instead, it allows the decoder to learn to just attend to some parts at each step of generation based on the input sequence and what has been produced so far (Montantes, 2019).

Attention is a mechanism that computes the importance or relevance of each input token with respect to the current decoding step. It allows the model to assign different weights or attention scores to different parts of the input sequence, highlighting the most relevant information for the current prediction.

2.3.2.1 Self-Attention

The self-attention mechanism is a key component of transformers that allows the model to attend to different positions within the same input sequence. Unlike traditional attention mechanisms that focus on the interaction between two different sequences, self-attention enables the model to capture dependencies and relationships within a single sequence. To attain a clear understanding of how self-attention works, consider the sentence "Bark is very cute and he is a dog" as an example. This sentence has 9 words or tokens. The positionally closest words to the word "he" are "and" and "is" the preceding and succeeding respectively. But these words do not provide any context for the word "he". Even though the words "Bark" and "dog" are more related to "he" from a contextual point of view. It seems that context is more relevant than proximity.

Assume that the same sentence is fed to a computer, in this case, each word is considered as a token t and each token is mapped to a word embedding V. The goal is to enhance the word embedding form V to Y which contains more context information by providing some kind of weighting or similarity. The concept of context relevancy is shown in Figure 16.

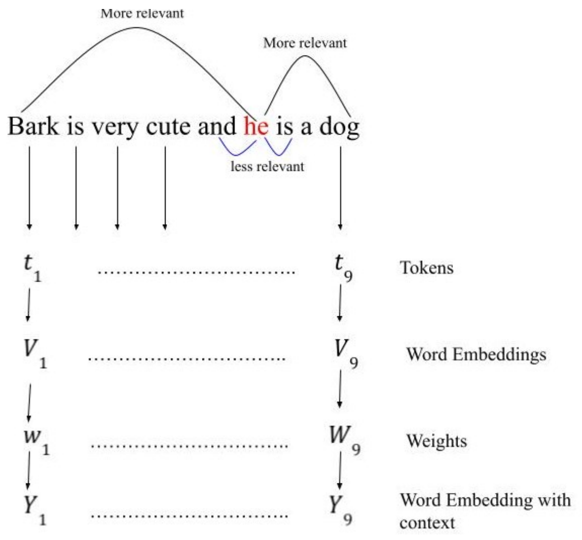

More relevant
More relevant
Bark is very cute and he is a dog
less relevant
t_1 .... t_9
Tokens
V_1 .... V_9
Word Embeddings
w_1 .... W_9
Weights
Y_1 .... Y_9
Word Embedding with context

Figure 16. The concept of context relevancy (Sarkar, 2022)

In an embedding space, semantically similar words are supposed to have similar embeddings. For example, the word "king" is more related to the words, "queen" and "royalty" than to the word "zebra". Similarly, "zebra" will be closer to "horse" and "stripes" than the word "emotion". Intuitively, the word embeddings will be multiplied by each other to obtain the weight vectors W. The initial embedding of the first word will be multiplied by the embedding of all words in the sentence. These weights are normalized to have a sum of 1.

$$V_1 V_1 = W_{11} \quad , \quad V_1 V_2 = W_{12} \quad , \; V_1 V_3 = W_{13} \quad \ldots\ldots\ldots\ldots \quad V_1 V_9 = W_{19}$$

Those weight vectors would be used in the calculation of new word embeddings to attain more context. Therefore, the weights are multiplied by the initial embeddings of all the words in the sentence.

$$W_{11} V_1 \; + \; W_{12} V_2 \; + \; W_{13} V_3 + \ldots\ldots\ldots\ldots + W_{19} V_9 \; = \; Y_1$$

$$W_{21} V_1 \; + \; W_{22} V_2 \; + \; W_{23} V_3 + \ldots\ldots\ldots\ldots + W_{29} V_9 \; = \; Y_2$$

.

.

.

$$W_{91} V_1 \; + \; W_{92} V_2 \; + \; W_{93} V_3 + \ldots\ldots\ldots\ldots + W_{99} V_9 \; = \; Y_9$$

W_{11} to W_{19} are all weights that have the context of the word V_1. so, when these weights will be multiplied with each word, those words are reweighted to the first word. This process will be repeated for all words in the sentence so, each word gains some context from every other word. It should be remarked that the order or proximity of the words does not influence each other. This approach of adding some context to the words in a sentence is known as Self-Attention (Sarkar, 2022).

2.3.2.2 Query, Key, and Values

There is an issue with the described scenario in the previous part which is it includes no trainable parameters in the formulation. It is evident that if some trainable parameters will be added to the system, the network learns some patterns and captures better context during training.

In the considered instance sentence "Bark is very cute and he is a dog", it would be observed that the initial embedding vectors V appeared 3 times in the calculation of final embeddings Y. These three occurrences of V can be referred to by the concepts of Query, Keys, and Values. The Query could be described as the current state of the network, Values indicate what the network is going to pay attention to, and Keys determine the degree of attention that should be paid to its corresponding value.

In self-attention, the input sequence is divided into three components: queries, keys, and values. These components are derived from the same input representation, but they are linearly transformed into different projections. Each position in the sequence generates its own query, key, and value vectors.

To make the concept of Query, Keys, and Values, assume that the goal is to extract all the words similar to the first word V_1. In this definition, V_1 is the Query. This query then will do a dot product with all the words in the sentence ($V_1$to V_9) which are addressed as Keys. The idea is depicted in Figure 17. consequently, the combination of Query and Keys results in the weights. In the next step, the weights will be multiplied with all the words which this time are referred to as Values in order to compose the final embeddings.

The context-based embedding that is calculated by the idea of Query, Keys, and Values as the self-Attention mechanism, would be formulated in the form of matrices to be able to be trained with a neural network. Accordingly, the Key vectors $V_1$to V_9would be multiplied by a matrix $M_k$of shape $k \times k$ (key matrix). similarly, the Query vector is multiplied with the matrix M_q, and the Value vectors are multiplied with the Values matrix M_v. all the values in these matrices can be trained by the neural network and provide better context than just using self-Attention.

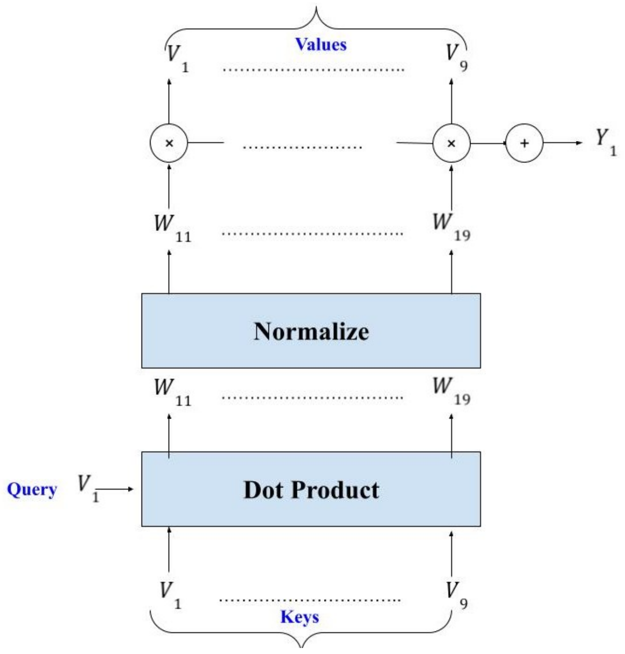


Figure 17. Query, Key, and Values concept in Self-Attention (Sarkar, 2022)

The idea of Query, Keys, and Values comes from database operations. If a database is considered a structured storage of key values, to retrieve a value v_i from the database based on query q and key k_i, some operations should be done. The query will be sent to the database to identify a key that corresponds to a certain value. Attention is supposed to be a process like the database operation but in a probabilistic manner.

$$attention(q,k,v) = \sum_i similarity(q,k_i) * v_i$$

The only difference between database retrieval and attention is that in database retrieval we only get one value as input, but in attention, we get a weighted combination of values. In the attention mechanism, if a query is most like key 1 and key 4, then both these keys will get the most weights, and the output will be a combination of value 1 and value 4. Figure 18 shows the process of calculating the final attention value from the query, keys, and values.

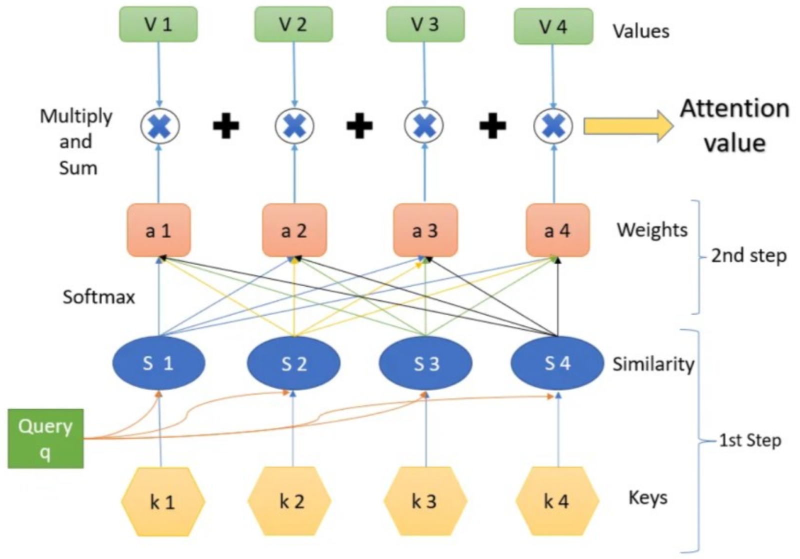


Figure 18. The process of calculating attention value from the query, keys, and values (Sarkar, 2022)

As is shown in Figure 18, in step 1 the similarity values will be measured using query and keys. Both the query and the keys are embedding vectors. Similarity *S* can be calculated using various methods. Some examples of similarity functions are formulated as follows:

$$s_i = f(q,k_i) = q^T k_i \qquad dot\ product\ method\ (T\ is\ transpose)$$

$$s_i = f(q,k_i) = q^T k_i/\sqrt{d} \qquad scaled\ dot\ product\ (d\ is\ the\ dimensional\ of\ each\ key)$$

$$\begin{aligned} s_i &= f(q,k_i) \\ &= q^T W k_i \qquad general\ dot\ product\ (query\ is\ projected\ in\ a\ new\ space) \end{aligned}$$

Step 2 is to find the weights a using *SoftMax* function. The similarities are connected to the weights like the fully connected layer.

$$a_i = \frac{\exp(s_i)}{\sum_j \exp(s_i)}$$

The last step calculates the weighted combination of the results of the *SoftMax* (a) with the corresponding value (V) (Sarkar, 2022).

$$attention\ value = \sum_i a_i V_i$$

The attention block would be developed as neural network representation which is illustrated in figure 19. The word embeddings are fed to some linear layers that do not have "bias" elements and just do matrix multiplication. Each linear layer is assigned to the query, keys, and values. This block can now be used in a neural network and is known as the Attention block. Multiple such attention blocks can be added to provide more context. And gradient backpropagating could be used to update the attention block (weights of keys, queries, values).

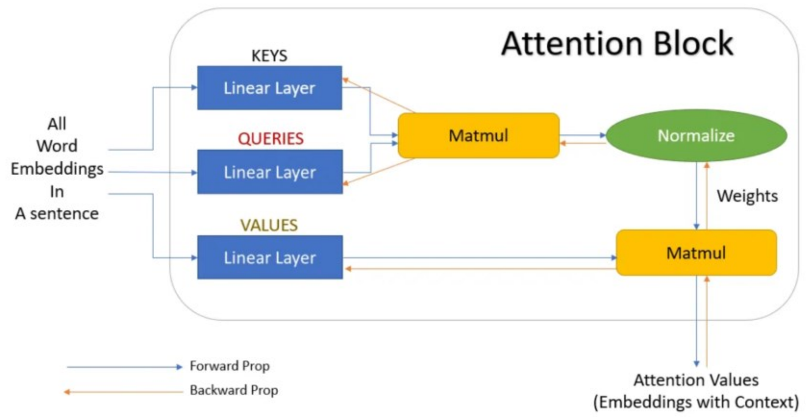


Figure 19. Attention block (Sarkar, 2022)

2.3.2.3 Multi-Head Attention

In the context of transformers, attention mechanisms play a vital role in capturing dependencies and modeling relationships within input sequences. Two common variations of attention are single-attention and multi-head attention.

Single attention refers to the use of one single attention head in the transformer model. In this case, the model computes attention scores between query and key vectors and applies the resulting weights to the corresponding value vectors, producing an attended representation. Single attention allows the model to focus on relevant parts of the input sequence, but it may have limitations in capturing diverse patterns and attending to multiple aspects simultaneously.

On the other hand, multi-head attention extends the attention mechanism by incorporating multiple parallel attention heads. Each attention head learns to attend to different patterns or aspects of the input sequence independently. Therefore, the model performs multiple attention computations in parallel, resulting in multiple sets of attended representations.

To make it clearer, consider the previously mentioned example sentence "Bark is very cute and he is a dog". In this example, the words "Bark", "cute", and "he" are semantically and grammatically corresponding to the word "dog" and give some further information about it. Only one attention head may not be able to correctly identify all these three words as relevant words to the word "dog". More attention heads, for instance, 3 heads could be added to the architecture to manage to capture more relevant words.

In order to develop a muti-head attention block, it is required to extent the block with more linear layers as the queries, keys, and values. These layers would be trained independently and in parallel. These multiple attention layers will produce multiple outputs that should be concatenated to give the final attention output. The architecture of multi-head attention is demonstrated in Figure 20 (Sarkar, 2022).

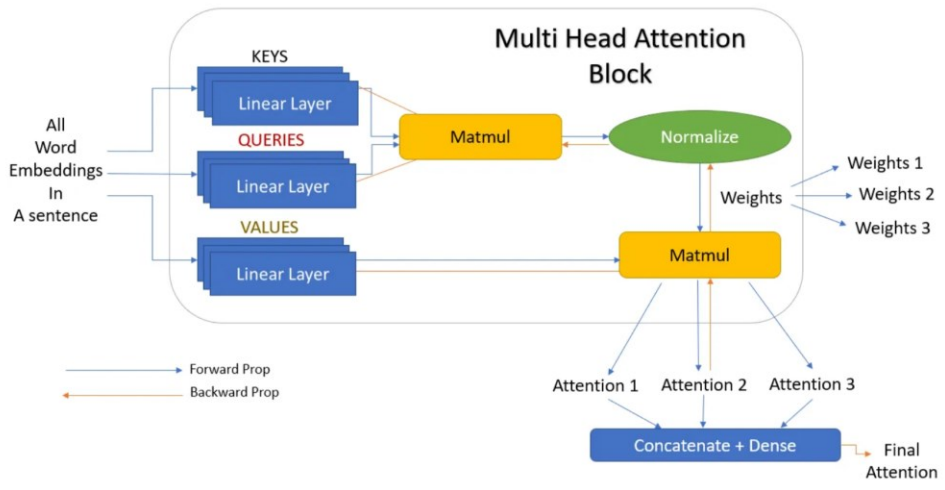


Figure 20. Multi-head attention (Sarkar, 2022)

Multi-head attention offers several advantages over single attention. First, it allows the model to capture different types of information and patterns in the input sequence. Each attention head specializes in attending to specific relationships or dependencies, providing a more comprehensive understanding of the data. By aggregating the outputs from multiple heads, the model can consider various perspectives and utilize diverse information when making predictions.

Additionally, multi-head attention enhances the model's ability to model long-length dependencies. Different attention heads can attend to different positions and capture relationships across a wide range of distances. This enables the model to effectively handle tasks that require long-term context and understanding of the input sequence.

Another benefit of multi-head attention is its ability to improve model stability. Since it may have a smaller number of layers than single attention to address the same number of positions. On the other hand, according to recent advancements in Deep Learning, training stability is not a problem anymore. So, deep single attention should be replaced with a multi-head attention block (Liu et al., 2021).

While multi-head attention offers advantages, it does come with increased computational complexity compared to single attention. Each attention head introduces additional parameters and requires separate computations, which can make the model more resource intensive. However, the benefits in terms of improved performance and representation power often outweigh the computational costs.

2.3.3 Transformers

Transformer was introduced in 2017 by Vaswani et al. as a neural network architecture that revolutionized the field of NLP also effectively influenced computer vision approaches. It has become the foundation for state-of-the-art models like BERT and GPT. Transformer architecture is solely based on the attention mechanism and the recurrence and convolution are entirely removed which resulted in parallelizing and requiring less time to be trained (Vaswani et al., 2017).

As has been discussed in the previous chapters, most competitive sequential models have an encoder-decoder structure. The encoder maps the input sequence to representation which is usually a space with lower dimensionality and then the decoder generates an output sequence using that representation. This process continues sequentially, at each step, the previously generated symbols will be fed to the network in addition to the input.

In Transformer architecture, similarity there are encoder-decoder stacks, but the input will not be processed sequentially. The encoder of the transformer can process the whole sentence in parallel, making it much faster and better than RNNs. This architecture is shown in Figure 21.

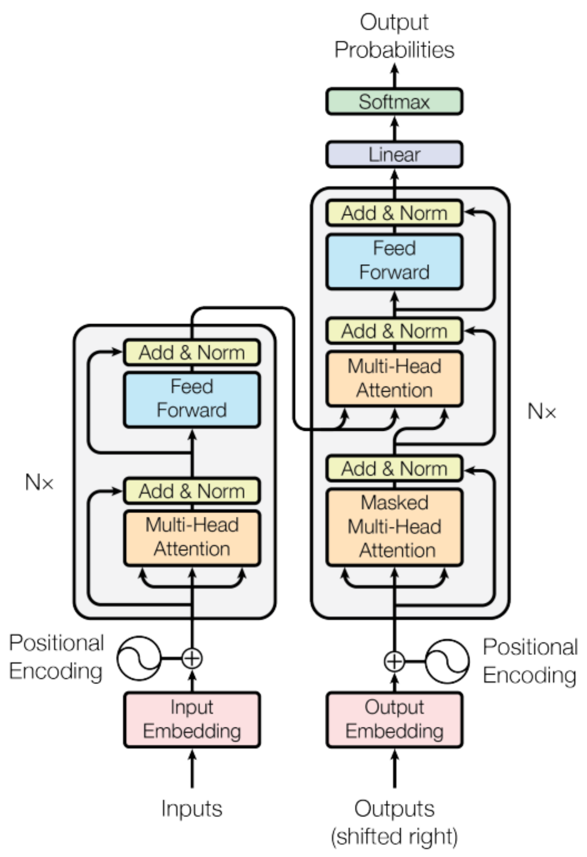


Figure 21. Transformer architecture (Vaswani et al., 2017)

2.3.3.1 Encoder

The whole input sentence is fed to the encoder through an ‘Input Embedding’ block which is followed by a ‘positional encoding’. Input Embedding is a module in the encoder that converts a word to its vector representation. In the embedding space, similar words have similar embeddings. The ‘Positional Encoding’ is essential to maintain a record of the position of each word in the sentence. Because the sentence is passed to the pipeline entirely not sequentially, it is needed to store the positional information. Without Positional Encoding, the model considers the input sentence like a bag of words with no meaning.

The idea behind Positional Encoding comes from the fact that a word in different sentences can have different meanings. For example, in the sentence “I own a cute dog.” the word ‘dog’ means a pet and appears in position 5 but in the sentence “What a lazy dog you are!”, it addresses being worthless and occurs in position 4 and it comes with the word ‘lazy’ that give context to it. Therefore, Positional Encoding would provide information based on the context and the position of the word in the sentence. The ‘Positional Encoding’ is added to the ‘Input Embedding’ before going through the attention network. So, both should be with the same dimensionality.

There are various methods to implement Positional Encoding. Vaswani et al. have utilized sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/1000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/1000^{2i/d_{model}})$$

Where pos is the position, and i is the dimension. Thus, each dimension of the positional encoding corresponds to a sinusoid.

The encoder is composed of a stack of N=6 identical layers producing an output of dimension $d_{model} = 512$ (Vaswani et al., 2017).

When the positional encoding is added to the input embedding, the results will be passed to the multi-head attention block. This block receives a vector including vectors representing words in the sentence having positional embedding. The multi-head attention computes the attention of every position with every other position of the vector.

During the attention computation, each word is considered as a "query" and matched with other words in the sentence referred to as "keys". The attention mechanism then combines the corresponding "values" in a weighted manner. This part of the process is performed using the Scaled Dot-Product Attention pipeline which is shown on the left side of Figure 22. In multi-head attention, multiple sets of values, queries, and keys are employed, allowing for multiple attention calculations that incorporate contextual information. These different attention results are combined to obtain a final attention value, which encompasses the context from all words and multiple attention computations. This approach proves more effective compared to using a single attention block alone. Multi-head attention block is depicted in detail in Figure 22.

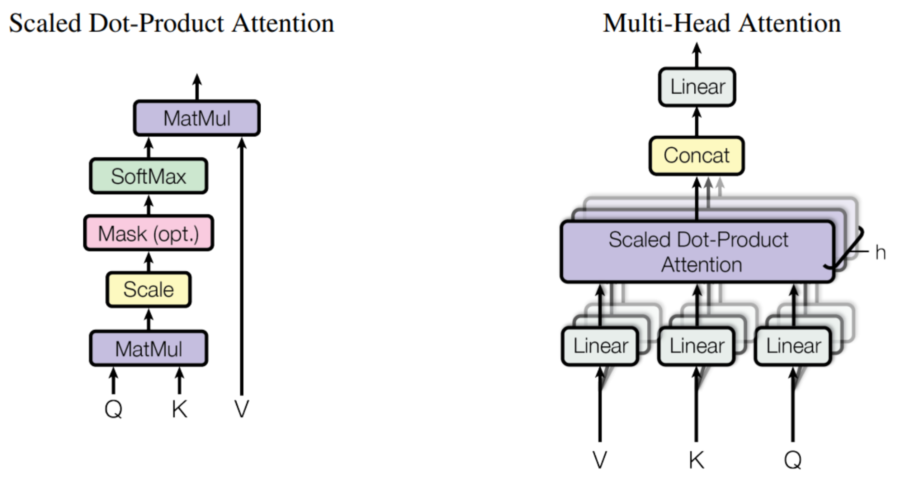


Figure 22. Multi-head attention in Transformer (Vaswani et al., 2017)

In Scaled Dot-Product Attention, the input consists of queries and keys of dimension d_k and values of dimension d_v. The results of the dot product of the query with all keys would be divided by $\sqrt{d_k}$ and then the weights are calculated by applying the softmax function.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Where Q, K, V are matrices of queries that would be passed to the attention function simultaneously packed together, packed keys and values respectively.

In the subsequent module, we encounter the 'Add & Norm' block, which involves taking the residual connection of the initial word embedding, adding it to the embedding derived from the multi-head attention, and subsequently normalizing it to achieve a mean of zero and variance of 1. This processed output is then passed into a 'feed forward' block, which similarly includes an 'add & norm' block at its output.

The entire sequence of multi-head attention and feed-forward blocks is repeated N times, where N represents a hyperparameter, within the encoder block (Sarkar, 2022 Vaswani et al., 2017).

2.3.3.2 Decoder

The decoder, with its own self-attention mechanisms, generates an output sequence based on the encoded representations. The output of the encoder which will be fed to the decoder is a sequence of embeddings including one embedding per position that contains the embedding of the original word at that position and information about other words learned by attention. Decoder attends to relevant parts of the input sequence and generates predictions step by step. This makes the

Transformer well-suited for tasks like machine translation, where the model needs to generate target sequences based on the source sequences.
Vaswani et al. proposed the Transformer primarily for machine translation tasks. For instance, the encoder takes in the English sentence and the decoder will translate it into French. Then they showed by experiment that it will be well generalized to other tasks. Depending on the application, the decoder is not necessarily required.
By considering machine translation from English to French task as an example, the decoder blocks perform as follows: the decoder block applies word embedding and positional embedding to the French sentence which has been taken in. Then, the self-attention block generates an attention vector for the French sentence. In the next step, the attention vector of French and English sentences will be compared which leads to word mapping from English to French. In the final layer, the decoder predicts the best probable translation from English to French. This process will be repeated till the entire input text is translated. Figure 23 shows the steps of the translation process in the decoder.
The decoder is composed of a stack of N=6 identical layers like the encoder. The difference between the encoder and decoder structure is that in the decoder there is a masked multi-head attention. The masking operation is needed in order to prevent the output from being dependent on the subsequent positions so, some values should be masked. The masking function can be formulated in the Attention method described in the following formulation.

$$Masked\ Attention(Q, K, V) = softmax(\frac{QK^T + M}{\sqrt{d_k}})V$$

Where *M* is a mask matrix of 0's and -∞'s (Sarkar, 2022 Vaswani et al., 2017).

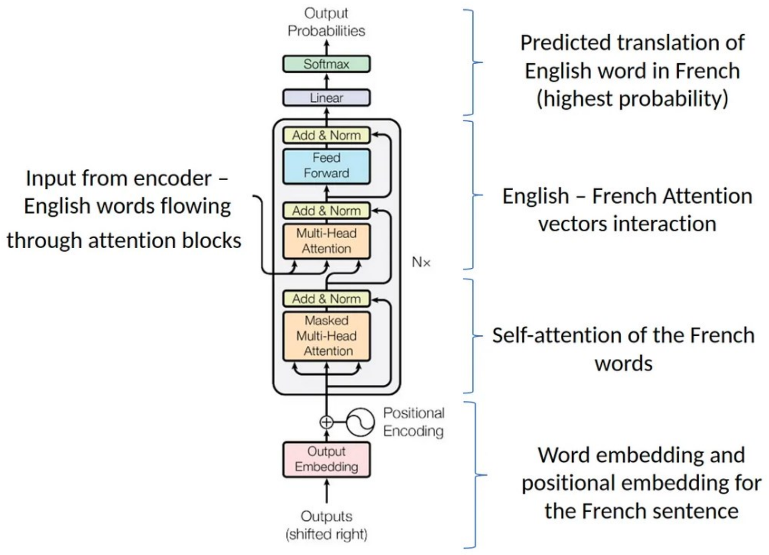

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)
Input from encoder – English words flowing through attention blocks
Predicted translation of English word in French (highest probability)
English – French Attention vectors interaction
Self-attention of the French words
Word embedding and positional embedding for the French sentence

Figure 23. Machine translation process in the Transformer's decoder (Sarkar, 2022)

One of the notable features of the Transformer architecture is its ability to handle variable-length sequences effectively. The model does not rely on fixed-length context vectors, as in traditional sequence-to-sequence models. Instead, it can adaptively attend to relevant parts of the input sequence, capturing the necessary information for each decoding step.

The success of the Transformer can be attributed to its self-attention mechanism, which allows for efficient processing of sequential data, its ability to capture complex patterns and dependencies, and its parallelizable architecture leading to computational cost reduction while still obtaining state-of-the-art results.

2.3.4 Transfer Learning

Training a deep neural network from scratch requires a giant quantity of data and consumes a lot of time and computational resources. Moreover, providing a large amount of labeled data for various tasks is burdensome. Transfer learning approach can be utilized to deal with these challenges.

Transfer learning is a technique to apply the already learned knowledge as a model to a new but related problem. It is a prevalent method in deep learning and specifically in Natural Language Processing (NLP) to improve learning efficiency significantly. It is a prominent approach because it is a solution to re-train the pre-trained large models on a comparatively small task-specific dataset to achieve satisfactory performance. For example, pre-trained deep learning models like BERT is fine-tuned for specific NLP tasks with relatively small amount of data.

Fine-tuning is a method of transfer learning in which the weights of the pre-trained model will be updated by training on a custom dataset. During the fine-tuning process, the weights of the entire network can be updated in the backpropagation step or just a subset of layers be adapted, and other layers will be frozen. The models pre-trained on large corpora are usually fine-tuned by utilizing the models' weights besides adding a task-specific layer which will be trained with a little labeled data in a supervised learning setting. Fine-tuning the full model can also be done and often yields better results, but it is more computationally expensive. It should be noted that a set of data with labels is the key requirement for fine-tuning a model.

Another approach in transfer learning is feature extraction. In this approach the captured knowledge in a pre-trained model is utilized to map the input data into a new representation containing contextual information captured by the applied model. These new representation features can be used to train a new model or directly applied to a downstream task.

In Figure 24 the difference between feature extraction and fine-tuning is depicted. According to the left side of the figure in the feature extraction strategy, the input is fed to the original model to be transformed into a new representation which is called features then, these features can be used to train a totally new model. With approach the captured knowledge in one model is transferred to another model without touching or updating the weights of the original model.

On the other hand, the right side of the figure illustrates the fine-tuning where the task-specific input is fed to the model to retrain the model for downstream task and create a new version of the model. The re-training can be done only on the last layer of the network by freezing the other parts. In this way the pre-trained (original) model is re-trained to fit the downstream task with less effort and resource consumption.

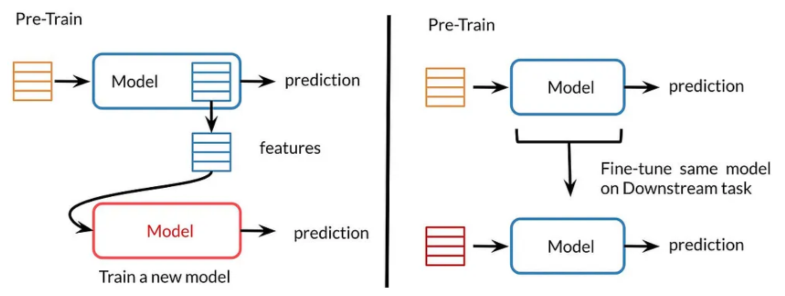


Figure 24. The difference between feature-based (left) and fine-tuning (right) (Yang, 2022)

2.3.5 Pre-trained Models

The scarcity of training data has been a major challenge in natural language processing (NLP). Most task-specific datasets have a limited number of human-labeled examples. However, deep learning-based NLP models benefit from larger datasets. To address this, researchers have developed techniques for training general-purpose language representation models using unannotated text on the web (pre-training). The pre-trained models can then be fine-tuned on specific NLP tasks with a small dataset, resulting in significant accuracy improvements compared to training from scratch.

2.3.5.1 BERT

BERT is a pre-training technique that trains deep bidirectional representations from unlabeled text. It captures both the left and right context in all layers, leading to a comprehensive understanding of language. By adding just one output layer, the pre-trained BERT model can be fine-tuned for tasks like question answering and language inference, achieving state-of-the-art performance without the need for substantial task-specific architecture modifications.

It can be trained quickly on a single Cloud TPU in less than 1 hour or GPU for a few hours. The release of BERT includes source code and pre-trained language representation models, showcasing impressive results on 11 NLP tasks, including question-answering on the challenging Stanford Question Answering Dataset (SQuAD v1.1) (*Open Sourcing BERT*, 2018).

BERT has been pre-trained with a dual objective. The objectives that were considered in training BERT include the 'masked language model' and 'next sentence prediction'. In the masked language model, some of the randomly chosen tokens will be masked from the output, and the model objective during pre-training is to learn to predict the original vocabulary id of the masked word based on its context. It should be noted that this pre-trained model is called deep bidirectional because both left and right contexts are taken into account to predict the masked token leading to a deep bidirectional Transformer architecture.

In addition to the masked language model, the 'next sentence prediction' objective is used to pre-train the model on text-pair representation. In other words, during the pre-training process, pairs of sentences are shown to the BERT model then, it is supposed to predict whether the second sentence follows the first one regarding the labels like contradiction, neutral, and entailment.

The BERT framework involves two steps: pre-training and fine-tuning. Pre-training entails training the model on unlabeled data with different tasks while fine-tuning initializes the BERT model with pre-trained parameters and fine-tunes all parameters using labeled data for downstream tasks. Despite sharing pre-trained parameters, separate downstream task-oriented fine-tuned models are formed. Figure 25 demonstrates the BERT framework for question-answering tasks. It consists of pre-training with both objectives that are followed by paired masked sentences shown on the left and fine-tuning for question-answering on the right side.

BERT's architecture comprises a multi-layer bidirectional Transformer encoder, it is an encoder-only Transformer. The model size is determined by the number of layers that could be Transformer blocks scripted as L, the hidden size as H, and the number of self-Attention heads A. The $\text{BERT}_{\text{BASE}}$ model architecture is formed by the following parameters $L = 12$, $H = 768$, $A = 12$ resulting in the number of *Total Parameters* = 110M. The $\text{BERT}_{\text{BASE}}$ parameters have been chosen in a way to have the same model size as OPENAI GPT for comparison purposes. There is a critical difference between GPT and BERT related to the masked model language concept. BERT Transformer uses bidirectional self-attention while the GPT Transformer uses constrained self-attention where only the left context is considered. The performance of the BERT model is checked with another set of parameters that constructed the $\text{BERT}_{\text{LARGE}}$ model with $L = 24$, $H = 1024$, and $A = 16$ resulting in *Total Parameters* of 340M (Devlin et al., 2019).

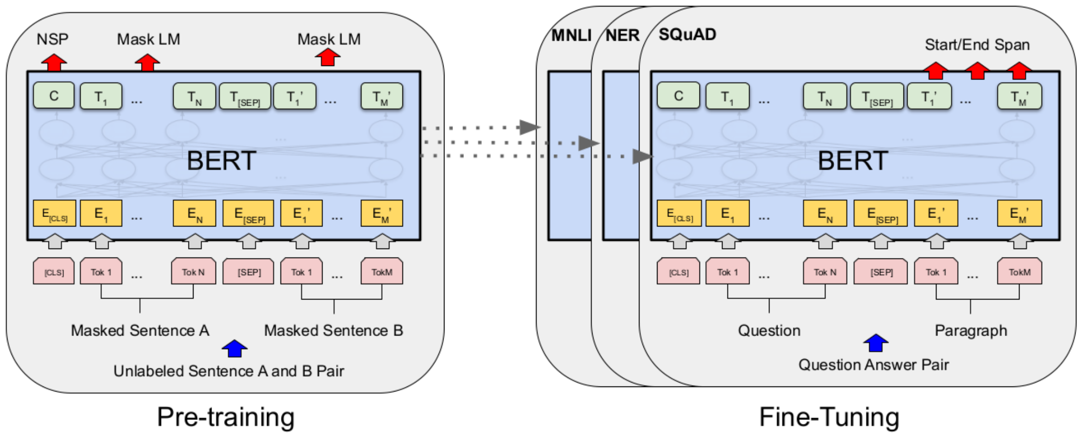


Figure 25. Pre-training and fine-tuning procedures for BERT (Devlin et al., 2019)

In the original paper proposed the BERT model by Devlin et al. based on experimental results, it is reported that the BERT model is effective for both fine-tuning and feature-based approaches. RoBERTa is another language model that improved upon the architecture of BERT by training the model for longer and longer batches, masking tokens randomly at each epoch, and modifying the objective function by removing the objective of predicting the next sentence.

2.3.5.2 GPT

The introduction of OpenAI's GPT models happened in 2018 by Radford et al. in a paper titled 'Improving Language Understanding by Generative Pre-Training'. It has had a remarkable influence on the NLP community. These powerful language models can effectively handle tasks like question answering, textual entailment, semantic similarity assessment, document classification, and text summarization without the need for supervised training which is extensively beneficial regarding the scarcity of labeled data. GPT models exhibit exceptional performance with minimal or no examples, often outperforming state-of-the-art supervised models.

The research shows that the Generative Pre-Training of a language model on a large corpus of unlabeled data followed by discriminative fine-tuning on specific tasks leads to significant performance improvement on various NLP tasks. So, the training procedure of GPT consists of two stages. The first step is to learn a large language model on a large corpus of text which is referred to as unsupervised pre-training. The next step is fine-tuning by adapting the learned model on labeled data of a discriminative task addressed as supervised fine-tuning.

The unsupervised pre-training stage can be formulated as a standard language modeling objective to maximize the likelihood of a sequence.

$$L_1(U) = \sum_i log\, P(u_i | u_{i-k}, \ldots, u_{i-1}; \theta)$$

Where $U = \{u_1, \ldots, u_n\}$, k is the size of the context window, and the conditional probability P is modeled by a neural network with parameter θ. These parameters are trained using stochastic gradient descent. A multi-layer Transformer decoder with multi-head self-Attention is used as the language model which is depicted on the left side of Figure 26.

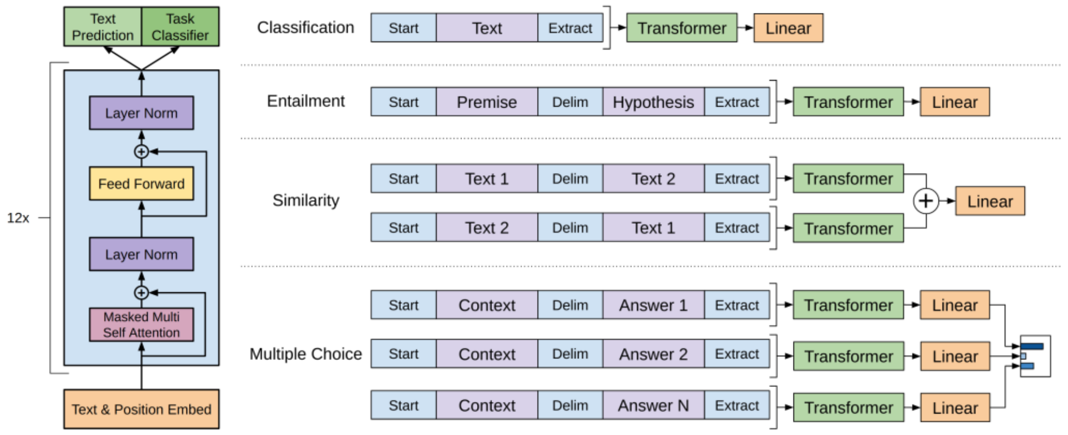


Figure 26. GPT Transformer architecture (left) and input transformation for fine-tuning on different tasks (right) (Radford et al., 2018)

When the model is trained, the parameters will be adapted to a supervised task. For supervised fine-tuning, it is assumed that there is a task-oriented labeled dataset C where each instance consists of a sequence of input tokens $x^1, \ldots, x^m$ with a label y. The input is passed through the pre-trained model to form the final Transformer block activation h_l^m. Then to predict the label y, h_l^m will be fed into the linear output layer with parameters W_y results in the following objective to be maximized.

$$P(y|x^1, \ldots, x^m) = softmax(h_l^m W_y)$$

$$L_2(C) = \sum_{(x,y)} log\ P(y|x^1, \ldots, x^m)$$

The objective function $L_2(C)$ is augmented with an auxiliary objective to improve the generalization of the supervised model and accelerate the convergence.

$$L_3(C) = L_2(C) + \lambda * L_1(C)$$

During the fine-tuning process, minimal changes were made to the model's architecture by transforming the inputs into ordered sequences. This transformation involved adding start and end tokens to the input sequences and using a delimiter token to separate different parts of the example. An illustrated example is shown on the right side of Figure 26. For tasks such as question answering or multiple-choice questions, multiple sequences were utilized for each example, incorporating the context, question, and answer in the case of question answering.

The GPT model follows the original Transformer and is trained as a 12-layer decoder transformer with a masked self-attention head with 768-dimensional states and 12 attention heads, so it is a decoder-only Transformer. Byte-Pair Encoding (BPE) vocabulary with 40000 merges has been used for tokenization. It just used the learned positional embedding instead of a sinusoidal version. The model has 117M parameters in total.

GPT, which is also referred to as GPT-1, showcased its superiority by outperforming specifically trained supervised models in various tasks. Additionally, it displayed notable zero-shot performance in various NLP tasks such as question answering, schema resolution, and sentiment analysis, thanks to pre-training. The success of GPT-1 emphasized the effectiveness of language modeling as a pre-training objective, enabling excellent generalization. The model's architecture enabled transfer learning and the execution of diverse NLP tasks with minimal fine-tuning (Radford et al., 2018) and (Shree, 2020).

The fascinating journey of the development of GPT pre-trained language models has been continued by OpenAI with a profound impact on various NLP applications. Followed by GPT-1, OpenAI released GPT-2 in early 2019. The development of the GPT-2 model is based on the idea of the ability of language models to perform downstream tasks in a zero-shot setting without any modification requirement in architecture or parameters. The idea was proposed by Radford et al. in 2019 in a paper called "Language Models are unsupervised multi-task learners".

The backbone of this approach is language modeling, particularly using self-attention architectures like the Transformer. It involves unsupervised distribution estimation from examples composed of variable-length symbol sequences. The joint probabilities over symbols are factored into conditional probabilities, enabling tractable sampling and estimation of $p(x)$ as well as any conditionals in the form of $p(x_{n-k}, \dots, x_n | x_1, \dots, x_{n-k-1})$.

On the other hand, the concepts of task conditioning, zero-shot learning, and zero-shot task transfer have been utilized in developing this multi-task model.

Learning a single task involves estimating a conditional distribution $p(output|input)$, but a general system should be able to perform multiple tasks even for the same input by conditioning on both input and the task $p(output|input, task)$ which is called task conditioning. A Language is a flexible tool for specifying tasks, inputs, and outputs as sequences of symbols to develop the task conditioning process. For example, a translation training instance can be written as the sequence (translate to French, English text, French text). Also, a reading comprehension training example can be written as (answer the question, document, question, answer).

Preliminary experiments showed that large language models can perform multitask learning without any task-specific module or parameters, although the learning process is slower compared to explicitly supervised approaches.

The main hypothesis in developing GPT-2 as a multi-task learner, Radford et al. 2019, is that a language model with enough capacity will start to learn and deduce the tasks depicted in natural language sequences, aiming to improve its predictive abilities, regardless of how those sequences are obtained. If a language model can achieve this, it essentially engages in unsupervised multitask learning. To examine this assumption, they evaluated the performance of language models in a zero-shot scenario across a diverse range of tasks.

One attractive capability of GPT-2 is its zero-shot task transfer. Zero-shot learning refers to a specific scenario where no examples are provided, and the model comprehends the task solely based on the given instruction. Unlike GPT-1, where sequences were rearranged for fine-tuning, GPT-2 introduced a different input format. This format expected the model to grasp the nature of the task and generate appropriate responses. For instance, in an English-to-French translation task, the model received an English sentence followed by the word "French" and a prompt (":"). The model is expected to understand that it needed to perform a translation task and provide the corresponding French counterpart of the English sentence. This approach aimed to simulate zero-shot task transfer behavior in GPT-2 (Shree, 2020).

GPT-2 has 1.5 billion parameters which is 10 times more than GPT-1 and a large vocabulary of 50257 has been used. The smallest model was used in the experiment (Radford et al. 2019) equivalent to the original GPT which has 12 layers, 768 model dimensionality resulting in 117M parameters in total, and the second smallest equivalent to the largest model from BERT (Devlin et al., 2018) containing 24 layers, 1024 dimensionality and the total number of parameters of 345M. GPT-2 has more parameters than GPT-1 including 48 layers, 1600 dimensional vectors for embedding, and 1542M parameters total.

It is shown by Radford et al. 2019, that the model is able to effectively handle a wide range of tasks in a zero-shot scenario indicating that high-capacity models, trained to optimize the likelihood of a suitably diverse and large text corpus, start acquiring the capability to perform numerous tasks without requiring explicit supervision (Radford et al., 2019).

The model is pre-trained on a very large corpus of data. It is pre-trained on the raw texts only, with no labeling which is called self-supervised training. It is trained to guess the next word in sentences where the inputs are sequences of continuous text of a certain length and the targets are the same sequence, just one token is shifted to the right. The model uses internally a masking mechanism to make sure the predictions for the token *i* only use the inputs from 1 to *i* but not the future tokens. in other words, in this type of training only the left context is used in predicting the next token. This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks. The texts are tokenized using Byte Pair Encoding (BPE) and a vocabulary size of 50,257. The inputs are sequences of 1024 consecutive tokens.

In continuation of OpenAI's journey in releasing new versions of GPT foundation models, GPT-3 published in July 2020. There were three models in that series with 1B, 6.7B, and 175B parameters. It has same architecture as GPT-2 but with modification to allow larger scaling. These models were superseded by the more powerful GPT-3.5 general models released in March 2022 and has 175B parameters. It can understand as well as generate natural language or code. Followed by that, GPT-4 was published in March 2023 as a set of models that improve on GPT-3.5 with capabilities of understanding as well as generating natural language and code. GPT-4 is more capable than GPT-3.5 in doing complex tasks moreover it is optimized for chat. The number of parameters in GPT-4 is undisclosed and is estimated to be 1.7 trillion. At the time that this research is done, GPT4 is the latest released version of GPT models.

2.3.5.3 DeBERTa

He et al. 2021 proposed the DeBERTa (Decoding-enhanced BERT with disentangled attention) which improves upon BERT and RoBERTa models through two novel techniques. The first technique involves disentangled attention, where each word is represented by two vectors encoding its content and position. Attention weights are computed using disentangled matrices based on content and relative positions. DeBERTa utilizes the absolute position vectors right after the Transformer layers but before the *softmax* layer for masked token prediction which is illustrated in Figure 27. Consequently, the position information is captured in all the Transformer layers, and it would be the absolute positions are the complementary information provided to decode the masked words. Because of that process, the decoding component in DeBERTa is called Enhanced Mask Decoder (EMD). As the right side of Figure 27 shows, EMD takes in two inputs including the hidden states from the previous Transformer layer *H* and any other necessary information for decoding *I,* e.g., absolute position embedding *H*, or output from the previous EMD layer.

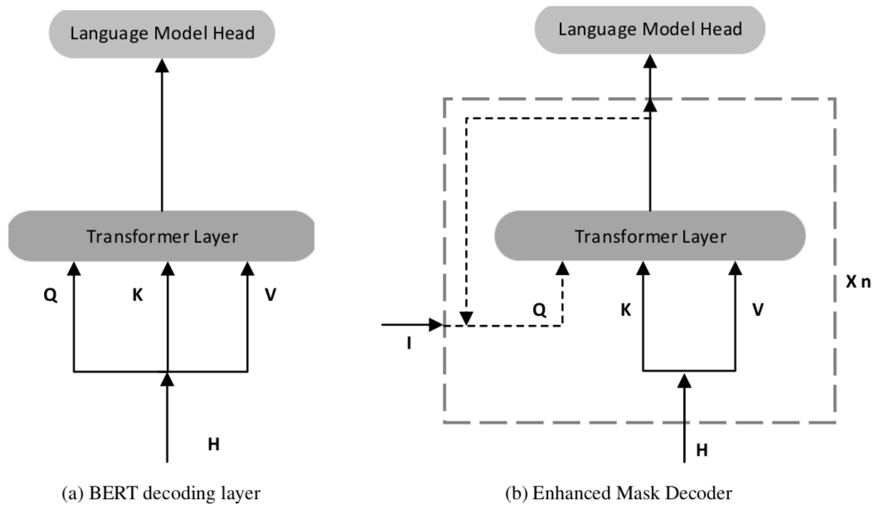


Figure 27. Comparison of the decoding layer in BERT (left) and EMD used in DeBERTa (right) (He et al., 2021)

The second technique incorporates an enhanced mask decoder that considers absolute positions during decoding to predict masked tokens in pre-training. Additionally, a virtual adversarial training method is used for fine-tuning to enhance model generalization. These techniques enhance the efficiency of model pre-training and improve performance in natural language understanding (NLU) and natural language generation (NLG) tasks.

By fine-tuning DeBERTa-XL on the MNLI dataset, the "*Microsoft/deberta-xlarge-mnli*" model is specifically tailored for tasks related to natural language inference. Natural language inference involves determining the logical relationship between pairs of sentences, such as whether one sentence contradicts, entails, or is neutral with respect to another. The fine-tuning process enables the model to specialize in understanding and accurately predicting these relationships.

DeBERTa with 1.5 billion parameters, denoted as DeBERTa$_{1.5B}$, consists of 48 layers with a hidden size of 1,536 and 24 attention heads. DeBERTa$_{1.5B}$ is trained on a pre-training dataset amounting to 160G, with a new vocabulary of size 128K constructed using the dataset (He et al., 2021).

2.3.5.4 MiniLMv2

Wang et al. 2021 introduced a task-agnostic compression method called deep self-attention distillation, building upon the work of MINILM (Wang et al., 2020). Self-attention relation distillation is employed to compress pre-trained Transformers. Multi-head self-attention relations are defined as scaled dot-products between query, key, and value vectors within each self-attention module which is demonstrated in Figure 28. This relational knowledge is then utilized to train the student model. The method allows for flexibility in the number of attention heads in the student model, unlike previous approaches. Fine-grained self-attention relations effectively leverage the interaction knowledge learned by the Transformer. Layer selection strategies for teacher models are thoroughly examined. Extensive experiments on compressing monolingual and multilingual pre-trained models demonstrate that the proposed models, distilled from large-size teachers like BERT and RoBERTa, outperform the state-of-the-art methods.

To transfer knowledge from a large model also known as Teacher to a smaller model called Student, knowledge distillation has been used widely as a promising way in pre-trained Transformer compression.

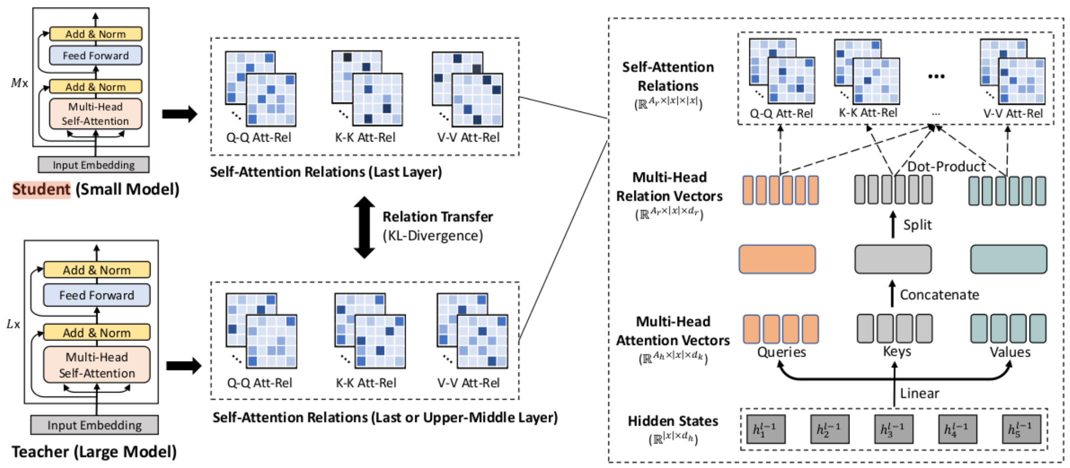


Figure 28. Overview of multi-head self-attention relation distillation (Wang et al., 2021)

Wang et al. 2021 reported the 6×768 model distilled from BERT$_{\text{BASE}}$ retains more than 99% accuracy of its teacher while using 50% Transformer parameters (Wang et al., 2021).

2.3.5.5 DistilBERT

According to what is mentioned in the previous part, knowledge distillation is a compression technique to train a smaller model (student) to reproduce the behavior of a larger model (teacher) or an ensemble of models. Sanh et al. 2020 proposed a triple loss function to transfer learning from the BERT model to a smaller general-purpose model called DistilBERT. The triple loss function combines language modeling, distillation, and cosine-distance losses.

Since variation in factors like the number of layers has a more major impact on computational efficiency than the hidden size dimension. The student, DistilBERT, has the same general architecture as BERT while the number of layers has been decreased by a factor of 2. In addition, the modern linear algebra framework is used to optimize the operation in Transformer layers to support a higher speed of computation.

This approach of knowledge distillation is leveraged during the pre-training phase leading to a general-purpose model instead of building a task-specific model. Sanh et al. 2020 showed that DistilBERT reduced the size of the BERT model by 40% even though 97% of the language understanding capability has been preserved and it is 60% faster. DistilBERT can operate on the edge and in situations where there are limited computational resources regarding the above-mentioned features (Sanh et al., 2020).

3. Semantic Search

Semantic search and Information Retrieval (IR) are closely related concepts in dealing with the task of finding and retrieving relevant information from a large collection of data even though they may handle it in different ways. Semantic search is a specialized approach to information retrieval that focuses on understanding the meaning of the user's query and the content of the documents to retrieve more contextually relevant results. Conceptual understanding, contextual relevance, and natural language understanding are the key aspects of semantic search. As it is discussed in Chapter 2, the NLP pipeline techniques has been leveraged to parse and understand the natural language used in the queries and documents, allowing for a deeper understanding of the content and concept.

3.1 Semantic Search Pipeline

When it comes to natural language processing, apparently textual data is the main data type which is categorized as unstructured data. Grainger et al 2021 explained that "text or any other data that doesn't fit a pre-defined schema ('structure'), is unstructured". Even though text data maintains the language rules and structure and conveys a tremendous amount of meaning and concept. Grainger et al. 2021 also stated that "there is much more structure hidden in unstructured data than most people appreciate. Unstructured information is more like hyper-structured information - it is a graph that contains much more structure than typical structured data". So, the golden goal is to harness this hidden power in order to create a semantic retrieval system.
The core operation in the semantic search pipeline is to extract the conceptual and contextual knowledge in the source into embedding vectors. The source corpora could consist of sentences, paragraphs, or documents that all are text-based and should be mapped into the embedding vectors. Then, at the search time, the search query must be embedded in the same vector space and the closest embedding would be found as the search results. The result is supposed to be a list of entries with high semantic correlation to the query which is ranked according to the similarity distance measurement (*Semantic Search — Sentence-Transformers Documentation*, n.d.-a). Hence, it could be deduced that semantic search complies with the pipeline of retrieval and ranking which are supplied by NLP and ML methods.
Semantic search approaches can be divided into two categories based on the length of the documents and the quantity of the content in the corpus. If the queries and the entries in the corpus are approximately of the same length and have the same amount of content, it will be considered

as a symmetric semantic search, for example, searching for similar questions. In symmetric tasks, the query and the entries could be potentially flipped.
On the other hand, if there is a short query like a short question or just some keyword and the goal is to find the long passage to answer the query, this case it would be interpreted as an asymmetric semantic search (*Semantic Search — Sentence-Transformers Documentation*, n.d.-b). according to the definition, the use case covered in this research is asymmetric semantic search.

3.1.1 Retrieval

Retrieval is the process of obtaining the most relevant objects in a collection to the asked information. The process of information retrieval (IR) relies on the query fed to the system. A query is a statement that expresses the required information. The search string in a web search engine could be a familiar example of a query. In information retrieval, a query may match several entries in the collection with different degrees of relevance so the results should be ranked according to the relevance measurement. Large language models have been applied to capture the meaning and contextual relations between the query and the objects to improve the accuracy of the retrieval process.

3.1.2 Ranking and Re-ranking

The output of the retrieval phase is a candidate set. The candidate items are ranked based on the degree of similarity. The similarity between them could be measured regarding various metrics that can be applied to determine the distance between vectors in the common embedding space. The possible options are cosine similarity, dot-product, and Euclidean distance. The process of retrieving ranked results based on similarity using the BERT model is introduced as Bi-Encoder. However, the retriever may not perform efficiently for a large set of documents and may return irrelevant documents. In this case, a re-ranker could be beneficial to improve the results. Cross-Encoder is one of the re-rankers implemented using the BERT model. (*Retrieve & Re-Rank — Sentence-Transformers Documentation*, n.d.). Besides that, an automatic text generation evaluation metric called BERTScore is introduced that is taken into account as re-ranking approach in this research.

3.1.2.1 Bi-Encoder

Bi-Encoder is a method for sentence pair scoring via generating sentence embedding in a vector space hence it can be used for applications like information retrieval, semantic search, or clustering. It produces an embedding for each sentence using a language model, for example, the BERT model. The sentences A and B, which result in the sentence embeddings u and v, would be passed to the language model independently. These sentence embeddings can then be compared using similarity functions like cosine similarity, dot product, etc. The Bi-Encoder procedure is illustrated in Figure 29.

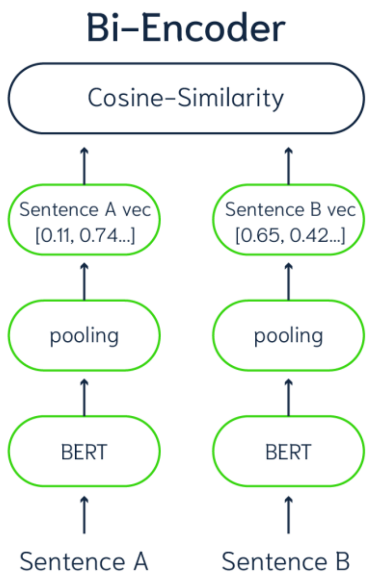


Figure 29. Representation of a Bi-Encoder model (Ham, 2022)

An advanced Bi-Encoder BERT model was presented by Reimers and Gurevych in 2019 called Sentence-BERT (SBERT) which is a modification of the BERT network employing Siamese (the same network for both sentences) with added pooling operation on top of it to extract fixed-sized embedding for input sentences and triplet network structures.

This modified architecture makes the SBERT derive semantically meaningful sentence embeddings suitable for tasks like large-scale semantic similarity comparison, clustering, and information retrieval in semantic search.

The SBERT is fine-tuned on NLI data which makes it perform efficiently on the Semantic Textual Similarity (STS) tasks. Fine-tuning with Siamese and triplet networks, consists of a triplet objective function considering an anchor sentence (s_a), a positive (s_p) and a negative sentence (s_n)

defined as $max(\left\|s_a - s_p\right\| - \left\|s_a - s_n\right\| + \epsilon, 0)$, generates embedding vectors that can be compared with cosine similarity (Reimers & Gurevych, 2019).

3.1.2.2 Cross-Encoder

Cross-Encoder model does not produce vector embedding for the data; instead, it is employed on top of the architecture providing embeddings and the retrieved candidate list. It operates like a classifier, which is shown in Figure 31, by passing a query and a possible document simultaneously to the Transformer network, for instance the BERT model, and producing a score indicating how relevant they are. If the BERT is considered as the core functionality, Cross-Encoder would be a fined-tuned version of it with one additional output layer acting as a classifier to determine how similar the inputs are. It is a sequence-level task because both input sentences are concatenated as a sequence but there is a special symbol to separate non-consecutive token sequence. In other words, as it is illustrated in Figure 30 [SEP] is used as special symbol for separating two sentences and [CLS] is applied for classification output.

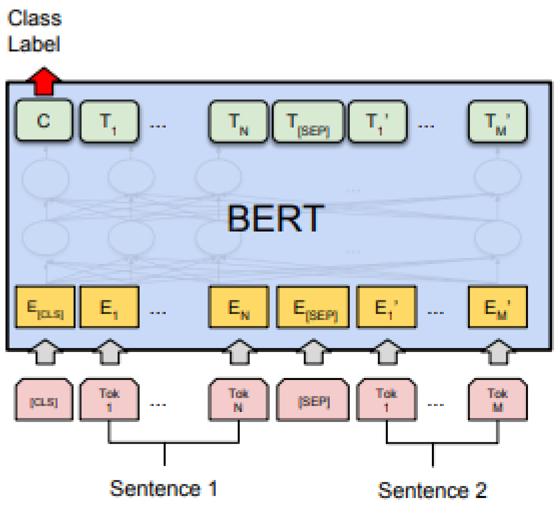


Figure 30. The functionality of Cross-Encoder as a fine-tuned version of BERT (Devlin et al., 2019)

A re-ranker based on a Cross-Encoder can substantially improve the results because they perform the attention mechanism across the query and the candidate document.

On the other hand, calculating the similarity score for many query-document pairs would be exceedingly slow. Therefore, the Cross-Encoder is usually used to score the candidate's retrieved documents by Bi-Encoder and re-rank the results to achieve higher performance (Ham, 2022) and (*Retrieve & Re-Rank — Sentence-Transformers Documentation*, n.d.).

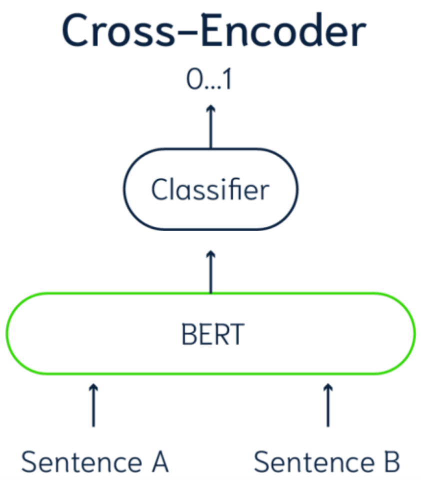


Figure 31. Cross-Encoder process (Ham, 2022)

In a Machine Learning pipeline, the tradeoff between high accuracy and speed is usually a crucial concern. The higher accuracy would result in more resource consumption and computational complexity. On the other hand, most use cases especially the user-faced systems like Information Retrieval are supposed to provide the user with accurate results with low latency. To consider both high accuracy and speed simultaneously, the combination of both retrieval and reranking methods including Bi-Encoder and Cross-Encoder would be effective to benefit the strength of both which is demonstrated in Figure 32. According to the studies, Bi-Encoder is computationally efficient but less accurate while Cross-Encoder provides more accurate results in the cost of consuming more resources (Chiu & Shinzato, 2022).

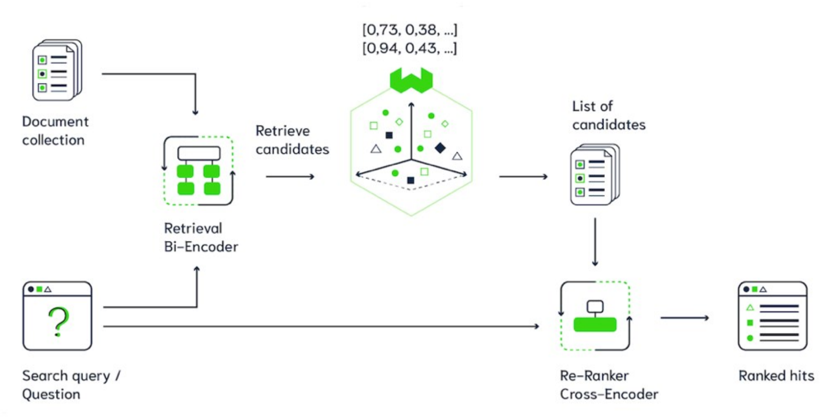


Figure 32. Combining Bi-Encoder and Cross-Encoder (Ham, 2022)

3.1.2.3 BERTScore

Automatic evaluation of natural language generation mandates comparison between the generated candidates and the reference sentences. Therefore, evaluation metrics need annotated reference sentences. If the reference sentence x and the candidate sentence $\hat{x}$ are given, the evaluation metric will be defined as a function $f(x, \hat{x}) \in R$. The better score is supposed to be more correlated to human judgment.

Text generation evaluation metrics would be categorized as *n*-gram matching, edit distance, embedding matching, or learning function. The metrics operating based on the *n*-gram algorithm like BLEU and METEOR, usually fail to match the paraphrases robustly which leads to poor performance on semantically related phrases that differ from surface form because those metrics depend on syntactic overlaps, in addition, *n*-gram models cannot capture distant dependencies and semantically order changes (Zhang et al., 2020).

Zhang et al. 2020 introduced an automation evaluation metric for text generation, called BERTScore, which computes the similarity between each token in the candidate and reference sentences. The score is calculated as a sum of cosine similarities based on the contextual embeddings of the sentences utilizing pre-trained BERT. Figure 33 shows the process of computing the recall metric of BERTScore consisting of BERT embedding capturing, pairwise cosine similarity defined as $\frac{x_i \hat{x_i}}{||x_i|| \, ||\hat{x_i}||}$, greedy matches, and importance weights. Research done on similarity measures demonstrated that rare words can be more indicative of similar sentences than common words so, the Inverse Document Frequency (idf) scores have been used to determine the weight importance of each token. The recall, precision, and F1 metrics would be calculated in the BERTScore model.

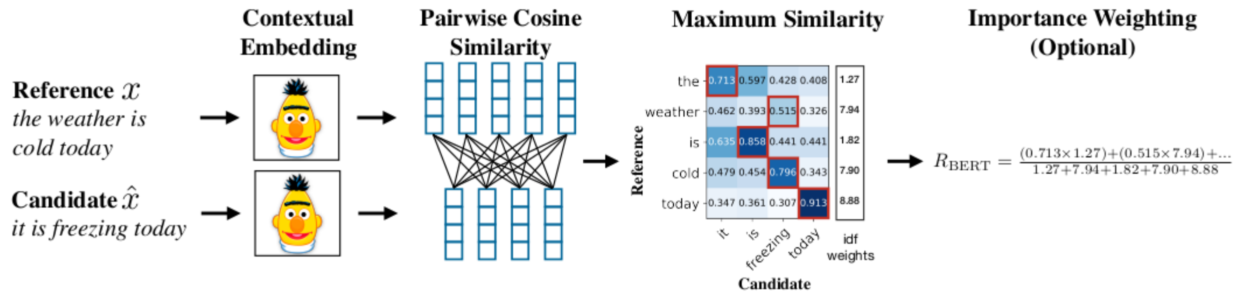


Figure 33. computation of the recall metric R_{BERT} (Zhang et al., 2020)

The BERTScore model is inspired by embedding-based and learned metrics but is not optimized for any specific evaluation task. Token-level computation in BERTScore weighs tokens differently according to their importance. So, it addresses the shortcomings of the other approaches.
Experiments done by Zhang et al. 2020 indicated that BERTScore associates more satisfactorily with human judgment and delivers adequate model selection measures rather than other existing metrics. It is recommended to use the DeBERTa model for text generation evaluation (Tianyi, 2019/2023) instead of the default RoBERTa model in the published paper.

3.2 Evaluation Metrics

To evaluate if the results of the search which could be a semantic search, information retrieval, or question-answering system, are satisfying a judgment list is required. A judgment list defines documents' relevance for a query (Turnbull, 2021). For instance, it can be a list of queries, the potentially relevant object that could be retrieved, and a grade of 1 or 0 assigned to each object which is depicted in Table 1 for a movie search engine as an example. As it is shown in the table, the movie "Star Wars: A New Hope" is a relevant object for the query "Satr wars" and attain a grade of 1 while the movie "Blair Witch Project" is irrelevant and obtained a grade of 0 which shows it is not a proper match for the query.
The judgment list, which is another term to describe the "ground truth" or "golden set", formed by obtaining annotations for all the documents in the search system is the key component to evaluate the performance of the search systematically rather than subjectively. Besides the judgment list another significant concept in designing a proper search relevance evaluation is to consider metrics that can validate the importance of the position of each object in the result list. In other words, occurring the most relevant object at the top level in the list should be more valued. With these requirements, the improvements of the search system would be evaluated without demanding for user test or A/B test.
The reliability of the relevance evaluation method is excessively dependent on the quality of the judgment list. Turnbull 2021 categorized the possible methods for providing a judgment list into three classes including Explicit Judgment, Crowdsourced Judgment, and Implicit Judgment.
In explicit judgment, direct feedback on being relevant or not will be obtained by recruiting evaluators that need to be selected based on some criteria like being an expert in the field and available budget in addition to, providing rating guidelines and training. The process will continue

after employing the experts by selecting the test queries, collecting the ratings, generating the score, and analyzing them. In this approach, the rater capacity directly affects the number of queries that can be tested also it should be regarded in the query selection procedure (Diedrichsen & Sierra, 2019).

Query	**Movie**	**Grade**
Star wars	Star Wars: A New Hope	1
Star wars	Star Wars: A New Hope	1
Star wars	Blair Witch Project	0
Star wars	A Star Is Born	0
Star trek	Star Trek Into Darkness	1
Star trek	Star Trek II: The Wrath of Khan	1
Star trek	Sense and Sensibility	0
Star trek	Forrest Gump	0
…		

Table 1. An example of a judgment list (Turnbull, 2021)

Another approach to creating the judgment list is the crowdsource via Mechanical Turk or other third-party firms.

The third class utilizes the implicit behavior of the user interactions like clicks, purchases, and conversations to generate the judgment list. In this case, a learning model should be created to rank the results of the search based on user satisfaction regarding the training data that they provide through their interaction with the search system.

It would be the primary description of an automatic relevance evaluation system that the core concept of it can be turned into three major questions that need to be answered as a designed model consisting of "What do users want from search?", "How do we turn that into training data?", and

"How do we know whether that training data is qualified?" (Grainger et al., 2021). Therefore, three steps should be followed iteratively to achieve the goal of finding the answers to the abovementioned questions. First, it should be understood what an ideal relevance is based on the user interactions so, a judgment list would be created from the signals coming from the user. Then deductive modifications should be created in the search development to move towards the ideal relevancies provided in the judgment list using training data. The final step in the iteration loop is to validate the model performance (Grainger et al., 2021).

Evaluating the Information Retrieval (IR) of any form from search to recommendation system should be done through mathematically defined measures to provide a paramount understanding of how accurately the IR system operates. These measures are split into two categories comprising online metrics and offline metrics. Online metrics would be feasible for calculation during the actual deployment and usage of the IR system through user interactions like what is described in Grainger et al. 2021 as implicit judgment. Offline metrics can be measured in the developing and test stages before being deployed in the actual deployment environment. Offline metrics are divided into two groups based on whether the order of the retrieved items impacts the metric score. The order-based metrics include Precision, Recall, and F1-measure while the non-order-based metrics consist of Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain (NDCG).

If we consider an IR system operation, the predicted results compare to ground truth provide four categories of items. The correctly identified items are called *TruePositive* which are predicted as relevant and have positive (being relevant) labels in ground truth also. The items that are predicted as irrelevant and are negative (being irrelevant) in the golden set, are referred to as *TrueNegetive*. Those items with positive labels that are predicted as irrelevant are called FalseNegetive and the items with negative labels that are predicted as relevant are addressed as *FalsePositive*. These terms are used in the definition of Precision, Recall, and F1-measure.

It should be mentioned that each of the metrics can be calculated on a special number of results that the IR system will produce which is referred to as *k*.

3.2.1 Percision

It measures the ratio of the correctly identified answers among the first k answers predicted by the model. In other words, it qualifies the number of the correctly identified item made of all positive predictions.

$$precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

3.2.2. Recall

It qualifies the number of the correctly identified item made of positive predictions that could have been made.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

Recall is an effortlessly interpretable evaluation metric with the indication that a perfect score means all relevant documents are retrieved. And with a smaller *k* achieving a higher score is more challenging. On the other hand, by assuming *k is* close to the number of all possible documents, a perfect score would be promising which is a disadvantage because it is deceptive. In addition, it is not order-based so the rank of the relevant document does not matter in this metric.

3.2.3 F1-measure

Precision and recall measure are the two types of metrics that could be made regarding only the positive class. Maximizing precision minimizes *FalsePositives* and maximizing recall minimizes *FalseNegatives*. Both *FalsePositives* and *FalseNegatives* can be minimized to reach better performance therefore F1-measure that combines Precision and Recall into a single metric would capture both properties. So, the more correctly predicted samples, the higher precision, and recall hence the higher F1-measure will be captured.

$$F1 - measure = \frac{(2 \times Precision \times Recall)}{(Precision + Recall)}$$

3.2.4 Mean Reciprocal Rank (MRR)

Mean Reciprocal Rank is an order-based metric which means that unlike Recall and Precision, returning the actual relevant results at rank 1 scores better than at rank k. Another attribute of the

MMR is that it takes into account the number of queries in the calculation. The MRR formulation is as follows:

$$MRR = \frac{1}{Q}\sum_{q=1}^{Q}\frac{1}{rank_q}$$

Where Q is the number of queries, q is a specific query, and $rank_q$ is the rank (position) of the first actual relevant result for the query q (Briggs & Carnevali, n.d.).
The main advantage of this metric is being order-based which is crucial for systems like question-answering but it just takes into account the first relevant item so, it is not suitable for use-cases that rather to return multiple options like recommendation systems or search engines (Briggs & Carnevali, n.d.).

3.2.5 Mean Average Precision (MAP)

Mean Average Precision is another order-based metric that considers precision, a relevance parameter R_k which is the relevant score of the k^{th} item (it could be 1 if it is relevant otherwise would be 0), and the number of queries in the formulation.

$$MAP = \frac{1}{Q}\sum_{q=1}^{Q}\frac{1}{m_j}\sum_{k=1}^{m_j}P(R_{jk})$$

Where m_j is the number of relevant documents for the query j, and R_{jk} is the rank at which the k^{th} relevant document would be found for query j (Briggs & Carnevali, n.d.) and (Glavaš, 2020).
It considers the order of predicted items, so it is suitable for systems with the demand of multiple retrieved items. The minor disadvantage of this metric is that it assigns binary relevance parameters (Briggs & Carnevali, n.d.).

3.2.6 Normalized Discounted Cumulative Gain (NDCG)

Normalized Discounted Cumulative Gain belongs to the order-based metrics categories and is attributed to the graded relevance of documents that is not included in previously introduced metrics in this chapter. The fact that different queries generally could have different numbers of relevant documents, is conceptualized in the definition of NDCG with the intuitive insight that the Discounted Cumulative Gain (DCG) should be higher for queries with more relevant documents. If Ideal DCG (IDCG) is defined as the maximal DCG score that any ranking can have by considering the specific number of documents that each query could retain, the definition of NDCG will be as follows (Glavaš, 2020):

$$DCG = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)} \qquad IDCG(k) = \sum_{i=1}^{|relevant|} \frac{2^{rel_i} - 1}{log_2(i+1)} \qquad NDCG = \frac{DCG(k)}{IDCG(k)}$$

It is a significantly used metric in the evaluation of search engines as it is order-based, optimized for highly relevant documents, and interpretable but the drawback is that there is more complexity in calculation regarding estimating the data-oriented relevancy score for each item compared to the other items.

Each of the abovementioned metrics could be employed in various situations according to their advantages and shortcomings.

Applying both online and offline metrics would be more beneficial to support the continuous integration and development of the designed retrieval system (Briggs & Carnevali, n.d.).

3.3 Data Collection and Preparation

The documents, the main source of knowledge for the retrieval system to seek through it to find the best matches, contain a representative textual model of the search domain. This thesis explored capturing the textual model and contextual relationships within the content of the documents using language models. Therefore, in this research a dataset consisting of a proper field of text information to derive the textual model is required.

Recommendation systems utilize algorithms that can process various inputs such as user preferences, user behavior, content, and other factors. These inputs are leveraged to automatically identify and match the most relevant content for the user which is identical to what a search system commits. Therefore, search and recommendation systems should not be split into separate silos as they are two sides of the same coin and both of them are different kinds of information retrieval systems (Grainger et al., 2021).

According to the similar operation of the search engine and recommendation system regarding delivering relevant results based on understanding what the user is looking for and the equivalent evaluation methods that can be applied to both, in this research, a publicly available dataset which is an inspiration in all these applications is selected.

3.3.1 Dataset

Wikipedia Movie Plots dataset with 34886 movies from around the world containing eight features for each of them as columns of the data frame. The columns consist of *Release Year*, *Title*, *Origin*, *Director*, *Cast*, *Genre, Wiki Page* (URL of the Wikipedia page from which the plot description is scraped), and *Plot*. It is a publicly available dataset on Kaggle[1]. The *Plot* field is the source of knowledge in the designed semantic search pipeline. It consists of the description of the movie plots that can be of various lengths from short to long text scripts. This dataset is an inspiration for the different tasks dealing with natural language processing including *Content-Based Movie Recommender* recommending movies with plots like those that a user has rated highly, *Movie Plot Generator* generating a movie plot description based on seed input, such as director and genre, *Information Retrieval* return a movie title based on an input plot description, *Text Classification* predicting movie genre based on plot description.

3.3.2 Data Cleaning, Transformation, and Understanding

The csv formatted dataset is uploaded to the Colab notebook environment from Google Drive and stored in a Pandas data frame. So, the Pandas library is used to clean the dataset and transform some fields of data into a proper format. The data frame associates semi-structured data since it contains some fields like *Release Year*, *Genre* as a category, *Origin* and so on that can be readily queried and interpreted in addition to the *Plot* as pure text that should be processed to query it.
Data understanding remains the crucial step that precedes the development of any data pipeline to produce quantitative data as prerequisites for analytics purposes. This step is essential in avoiding unexpected problems during the next phase, data preparation, which is typically the longest part of any data science project.
In this stage, the research proceeded by identifying key characteristics, such as data volume and the total number of variables in the data, and comprehending the problems with the data, such as missing values, inaccuracies, and outliers. The data understanding procedure is directed into the following steps:

- Removing the duplication resulted in a reduction of 4525 in the total number of items.
- Reindexing the data frame by considering the *Title* field as the index element
- Scanning if there is any item with a Null value in columns.

[1] Wikipedia Movie Plots | Kaggle

- Dropping the rows of the data frame that contains a *Null* value in *Cast* column or '*Unknown*' value in *Genre* column.
- Cleaning up the *Genre* column using the *spaCy* library for NLP

The results of the scanning for Null are summarized in Table 2 which shows there is 1302 item that with no information about their cast so, those items are dropped due to missing value.

Column name	**Number of rows with *Null* value**
Genre	0
Origin	0
Director	0
Release Year	0
Cast	1302
Plot	0
Wiki page	0

Table 2. Scanning for Null values in columns of the data frame

The *Genre* field conveys information about the context of the movie, so it is more examined to be well formatted. It is also a categorial label utilized to split the data frame into smaller sub-frames to be processed in batches in order to cope with the limited memory and processing unit available for the current research.

The data frame contains 2094 unique Genres, and 5540 items hold an *'Unknown'* Genre value which is a kind of miss value so, specified rows are dropped. The dataset includes 31816 items after dropping those rows with *Null* and *'Unknown'* values in *Cast* and *Genre* columns respectively. The distribution of the data regarding Genre is demonstrated in Figure 34 depicting that the dataset is biased toward the drama and comedy genres which should be considered in analytics.

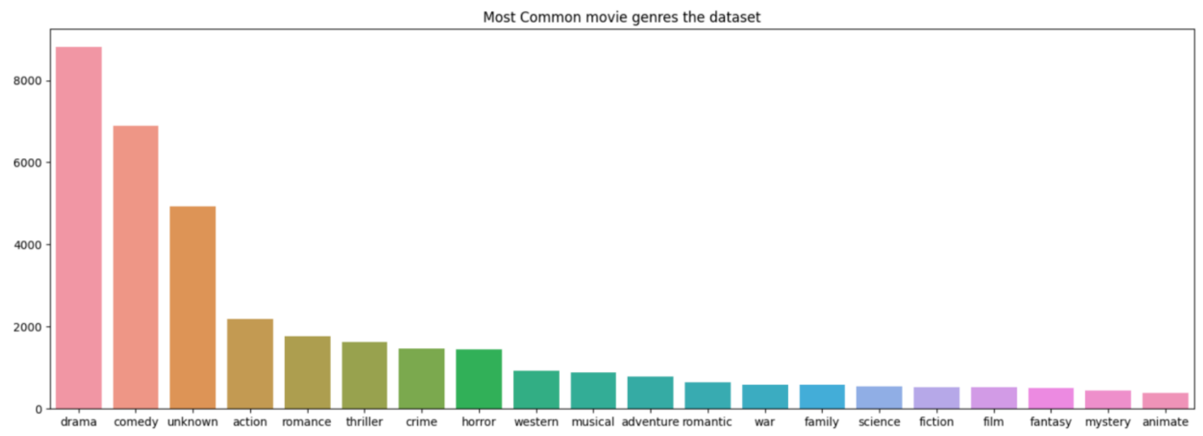


Figure 34. Genre distribution of movies in the data frame

It is discovered that there are some genre labels representing the same group but described with different words or phrases for example 'romance' and 'romantic' are grouped separately therefore, clean up function applied on that column of the data frame to mitigate the inaccurate diversity in genres. Besides, text cleaning and transforming compound genre's *string* into a list of genre labels are executed on the *Genre* values for instance, 'action thriller' is mapped into ['action', 'thriller'], and this new kind of genre description is accumulated to the columns as a new feature called *List_Genre*.

The cleaning job consolidates the descriptive and statistical analysis to produce a more accurate dataset which would be the bedrock of the realistic outcomes. One of the diagnoses originated from the analysis is to identify the outliers. The cleaning procedure followed up with recognizing and mitigating the outliers by grouping the movies by their *Genre* values after textual cleaning up. It is observed that there are some groups with very a small number of members compared to the other groups hence the rows belonging to those kinds of *Genre* are dropped to decline the outliers by considering 50 as the threshold of the number of movies in each group. The significant effect of the cleaning is restated in a table in Appendix B which shows the distribution of the data based on Genre.

The danger of misinformation is the fact that should be noticed in collecting and analyzing data. By reviewing the data more precisely regarding the *Genre* feature, it is found out that there is a group with the label of 'Animation' as *Genre,* but the fact is that 'Animation' is not included in movie genres. It is a type of medium and art form that can be done in any genre. There are more samples of this type of misinformation which is avoided by revising the label to 'unknown'. It emphasizes the importance of the data understanding phase to prevent the mistaken conclusion in analyzing the produced data based on the employed dataset. For instance, in the Text Classification task in which the movie *Genre* will be predicted corresponding to the *Plot* description, the preciseness of the conceptualized information in the Genre would be particularly substantial. Accordingly, misinformation will deceive the model and mislead it into incorrect predictions. However, in some cases recognizing the *Genre* based on the *Plot* is not straightforward even for humans e.g., categorizing a Plot into fantasy or fiction because of a slim boundary in the definition.

3.3.3 Data Labeling

Three methods are proposed and examined to approximate the human judgment to settle the scarcity of the annotated list of documents as being relevant to the specific query or not.
The ground truth is required to be able to articulate how well the workflow performs. The BERTScore and Cross-encoder are used to re-rank the candidates suggested by the Bi-encoder to determine the most relevant documents to each query and to calculate the evaluation metrics.
It is essential to choose a unique field of the dataset as the matching field in MRR calculation (as a common feature between the candidates and the judgment list), it could be the title of the movies, or a unique number assigned to each item (for example the Tile can be defined as the index in the data frame to be searchable by its values. This unique id is assigned properly because the duplications are dropped in the data cleaning phase by resetting the index in the Pandas data frame. Suitable index assignment results in more efficient retrieval.
Each introduced method requires specific parameters to be set based on the method requirements. These method-specific parameters are considered in the experimental investigation as dependent variables so, multiple assigned values for them are examined to discover how will affect the results.

3.3.3.1 Method 1

Annotating the whole collection of documents as a complete relevance judgment is not usually feasible even though it can be done for a small subset of the collection. However, for most queries, only a small subset of items is relevant, and a perfect retrieval system is supposed to rank those relevant items in the top position in the result list. Therefore, it would be concluded that just the smaller subset of the candidate list should be annotated. The proposed Method 1 and Method 2 are initiated by this idea. So, the re-ranking models, Cross-Encoder and BERTScore, are utilized to re-rank the candidate list and collect the intelligence deriving from both models to decide about the labels automatically. The shortcoming of this approach is that the IR engine may not operate perfectly consequently, some relevant documents may not appear in high-ranked positions and not be included in the candidate list so, an important part of the information would be ignored (Glavaš, 2020).

Method 1 leverages the similarity score calculated upon the pairs of queries and the candidates that are returned by the Bi-encoder. Section 3.1.2.2 describes the Cross-Encoder and Section 3.1.2.3 explains the BERTScore as a similarity score and text generation evaluation metrics utilizing pre-trained language models. According to knowledge exported from Section 3.1.2, those candidates will be re-ranked independently based on Cross-Encoder and BERTScore. A specific number of each re-ranked list will be selected as a new candidate list therefore, there will be two separate candidates list, one is re-ranked with BERTScore and the other one is re-ranked by the Cross-Encoder. The number of chosen items in these re-ranked lists is the parameter of this method. Then, the intersection of both re-ranked lists will be considered as the final relevant items. To be specific, it is the final judgment list produced by Method 1 which is illustrated in Figure 35. Selecting a proper number of candidates to be re-ranked is crucial from two perspective. First it depends on the use case for example, if the retrieval system is part of a recommendation system, or search engine returns up to even 30 candidates might be effective but if it is a question-answering platform, it could not be a long list. The second perspective arises from generalization which means by considering more candidates it would be more probable to achieve a better approximation of human judgment.

It should be mentioned that each *Plot* will be split into chunks based on an empirically determined number of tokens, which is set to 50 based on the experimental findings in this research so, the similarity score in Cross-Encoder and BERTScore is calculated for the pairs of each query and all chunks of each *Plot*. In other words, the re-ranked list is a collection of items that retain the top high-scored chunks.

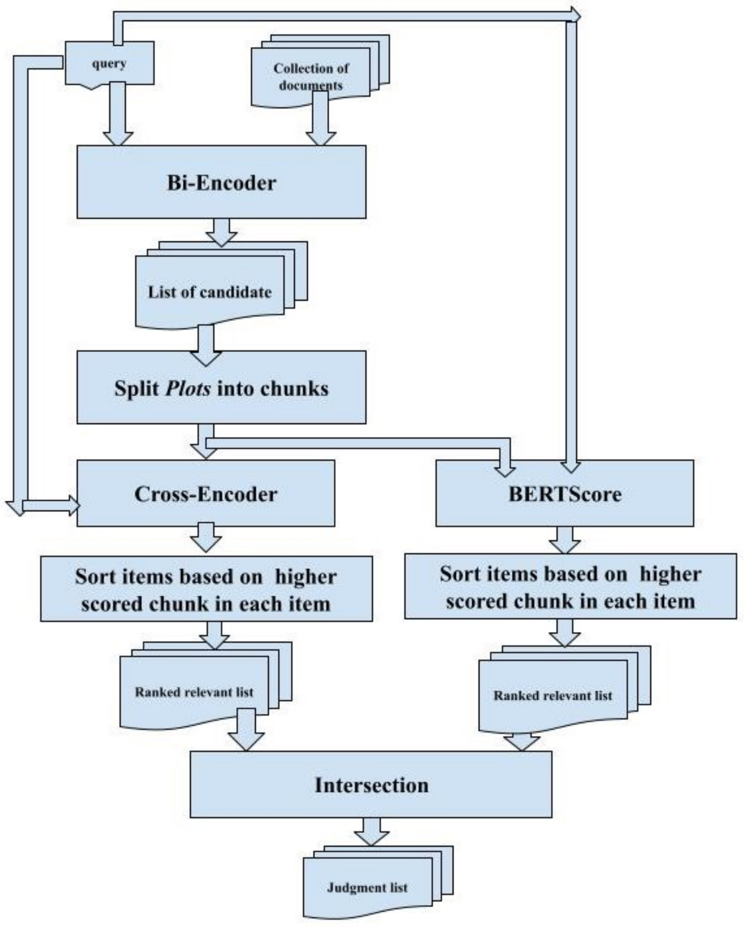


Figure 35. Creating a judgment list with Method 1

The values are set, equal to 5 and 30, for the number of selected candidates in the experimental investigation, and the results of each set are reported in the Result and Analysis chapter. It should be noticed that the total number of candidates that is returned by the Bi-Encoder is 50 which is considered in choosing the values 5 and 30 for the selected number of candidates.

If there is no common item between the Cross-Encoder re-ranked list and BERTScore re-ranked list, then the union of the top half items of each re-ranked list will be selected as the final judgment list.

The advantages of Method 1 consist of the evaluation of the performance of BERTScore and Cross-Encoder as re-rankers by the MRR metric and utilizing the collective knowledge of both models to decide on labels. On the contrary, the drawbacks are that if the Bi-Encoder does operate ideally, some of the actual relevant documents may not be included in decision-making, calculating the similarity between each query and all chunks of each *Plot* with a Large Language Model will be computationally expensive. The last disadvantage is that it considers only one language model in Bi-Encoder.

3.3.3.2 Method 2

The core idea behind Method 2 is the same as Method 1. Correspondingly, Method 2 considers the chunks of each *Plot* in the candidates' list as well and utilizes the similarity score calculated for the pairs of each query and all chunks of each *Plot*. in Method 1 the higher-scored chunk stood as the criterion to choose a Plot as the better match to be re-ranked in the top position by the Cross-encoder or the BERTScore. However, in Method 2, the frequency of the high-scored chunks belonging to the specific Plot is the measure for selecting an item to be ranked in a higher position. In other words, for each query, all the chunks of all candidate Plots will be considered in a flattened list. Then a threshold can be applied to choose for example only the first *k* number of items in that flattened list of chunks. In the next step, the selected chunks will be grouped by the movies that they belong to. The list of those movies (items) is the re-ranked list. This process will be repeated for both re-rankers. So, there will be two re-ranked lists of items and the intersection will be the final judgment list. This procedure is illustrated in detail in Figure 36. The number *k* is one parameter in this method that needs to be chosen by experimental study and there is a threshold on the number of members belonging to each group to be competent to be ranked in top position. This threshold is set to 10 based on the empirical assessment.

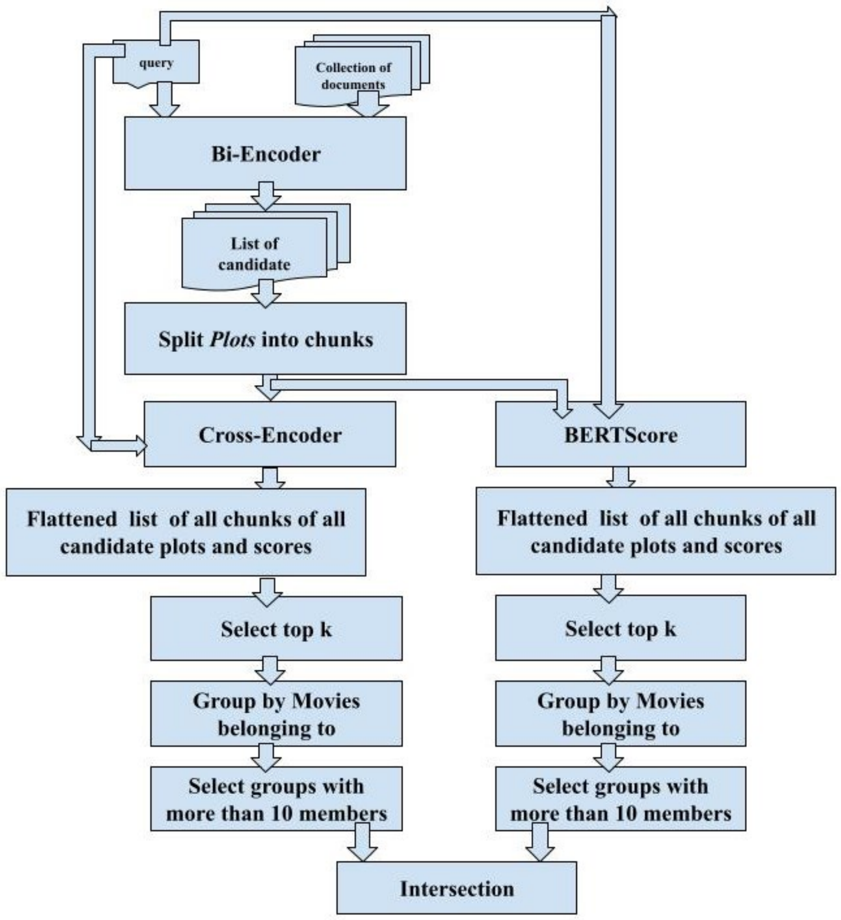

query
Collection of documents
Bi-Encoder
List of candidate
Split *Plots* into chunks
Cross-Encoder
BERTScore
Flattened list of all chunks of all candidate plots and scores
Flattened list of all chunks of all candidate plots and scores
Select top k
Select top k
Group by Movies belonging to
Group by Movies belonging to
Select groups with more than 10 members
Select groups with more than 10 members
Intersection

Figure 36. Creating a judgment list with Method 2

This method has the same benefits as Method 1 besides considering more probability for the plots with more than one high score chunk to appear in the top position. It materializes the fact that there may be some Plots that obtain a higher score for most of their chunks rather than just having one or a few chunks as relevant elements. So, this method does not have to choose necessarily a chunk per *Plot* (movie).

The penalties are the same as Method 1 which could be expressed succinctly as considering only one language model in Bi-Encoder, consumes a lot of resources, and ignoring some information depends on Bi-Encoder performance.

It should be noticed that re-ranking makes better results in the cost of more calculation, but it is better to do all this calculation for a subset of documents (candidates) instead of all documents.

3.3.3.3 Method 3

The Information Retrieval systems are usually evaluated in order to compare the performance of various models or different versions of the same model for a special application or task. Regarding this fact, it could be a good idea to leverage all the knowledge that will be provided by those models or variations of the model in a pooling method (Glavaš, 2020).

According to the pooling idea in addition to the point that this research explores the performance of two different models, Sentence-BERT (SBERT) and GPT-2, Method 3 resembles human judgment by constructing a union of the suggested judgment lists created independently by both models.

In this method, the risk of missing some relevant documents in the candidate list will be lessened because there is a chance for a document to be highly ranked and as a result be included in the candidate list with one model if it is not selected with the other one.

The empirical results of Method 1 and Method 2 for each of the models, Sentence-BERT and GPT-2, indicate that Method 1 applied to this research data performs better for Sentence-BERT even though Method 2 is the winner for GPT-2 which is discussed in more detail in Results and Analysis chapter. Therefore, as Figure 37 demonstrates, the union of the judgment list created by applying Method 1 using Sentence-BERT in Bi-Encoder and the judgment list produced by Method 2 by GPT-2 as the Language Model in Bi-Encoder.

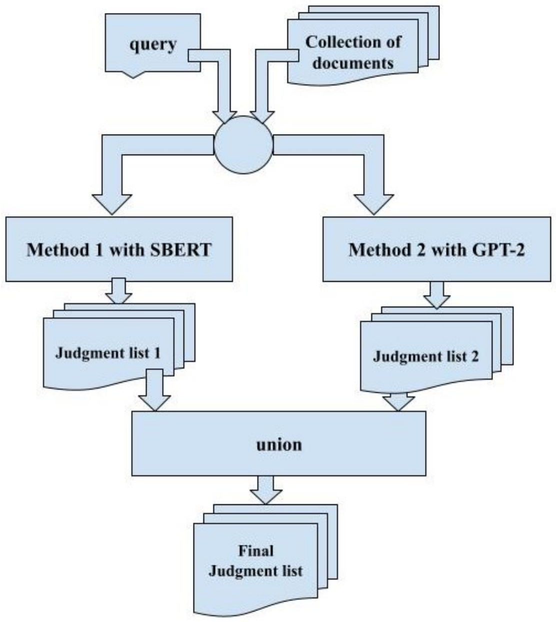


Figure 37. Creating a judgment list with Method 3

As it is discussed earlier Method 3 mitigates the danger of neglecting parts of relevant items, but it will happen in charge of more computation. Even though it combines the captured contextual information from both models which is supposed to be a kind of collective intelligence that may lead to more generalized and precise task-oriented judgement.

3.3.4 Query Attributes

Query plays a paramount role in an Information Retrieval system of any type, semantic search engine, recommendation system, question-answering, or retrieval text generation. The retrieved information will be picked from the collection predicated on the degree of the resemblance between the captured context from the query and the items in the collection. Correspondingly it should represent some attributes associated with being qualified to evaluate the performance of the model in retrieving the most relevant documents.

For the purpose of this research, a list of 7 synthetic natural language questions is organized regarding some measures of competency. The competent query should be navigational which means it should provide some hint to the system to look for relevant information for instance it may encompass some keywords in addition it would be related to at least one category of available data in the collection according to the designer's understanding about the dataset. It should be mentioned that in this research, the designer's role is taken by the researcher. In the current research, each query and its potentially relevant documents should belong at least to one movie genre.

The employed dataset was used in another research for creating a semantic search with SBERT and two of the asked queries there, were selected to be included in the test set of this research to provide a pivot for comparison (Verma, 2021).

Another consideration in designing the test queries is following the same intent as the borrowed queries from other research and articulating them in other words. With this criterion, the system will be checked for its capacity for Natural Language Understanding. In the same regard, some questions are designed in a way to navigate to the same genres and context to inspect the competency of the model in capturing the intent and context when it is expressed implicitly and can be compared with the queries that conveyed the intent explicitly through the keywords.

Those qualifications resulted in the following set of queries:

1. Artificial intelligence based action movie

2. Science fiction movie showing the future of the world.
3. Films about time traveling.
4. A movie about romance and the pain of separation
5. An action movie about revenge against family murder
6. Comedy movie that contains time travel fantasy

4. Implementation

4.1 Workflow

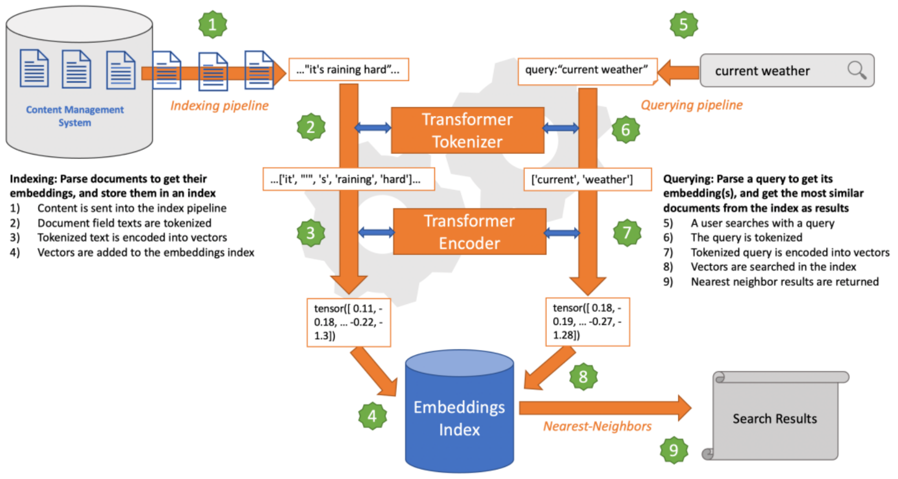
The text field in the current dataset, *Plot* of each movie, is the main content to provide the required knowledge from the unstructured data to be stored in a kind of index to be searchable. This process of extracting knowledge from the text encompasses two phases: preprocessing including the tokenization and mapping into embeddings. In the preprocessing phase, the text is analyzed using the spaCy library to split it into tokens, count the number of tokens, and strip out the noise like stop-words and special characters. It should be mentioned that stemming, and lemmatization are executed for the *Genre* field as a text string. The second phase applies machine learning approaches, specifically language models, to the text to derive the context of linguistic representations in the corpus.

In addition to capturing the semantic intent and meaning of the text field of the dataset by embeddings, the intent of the query needs to be understood by the same language model through generated embedding. In other words, the query should be converted into embedding also under the same setting.

After embedding generation and creating an index of them, any time a query is fed to the system, it will be converted to embedding. Then the system will look for the most similar embedding to the input query which is referred to as *nearest neighbors*. The similarity is calculated by distance measurements in dense vector spaces like cosine similarity, dot product, and so on. Figure 38 demonstrates an architecture of semantic embedding using the Transformers for comprehending the contextual intent and meaning of the contents as well as the query. What is shown in the figure can be summarized in 5 steps as follows:

1. Get the embeddings for all the Plots in the dataset.
2. Create an index with the embeddings.
3. Get the embeddings for a query.
4. Search the index.
5. Show the nearest neighbor results.

This workflow is the heart of this research configuration. The research questions are explored to be answered regarding this assumption. The code snippet of this procedure is attached in Appendix C.



Indexing: Parse documents to get their embeddings, and store them in an index
1) Content is sent into the index pipeline
2) Document field texts are tokenized
3) Tokenized text is encoded into vectors
4) Vectors are added to the embeddings index

Querying: Parse a query to get its embedding(s), and get the most similar documents from the index as results
5) A user searches with a query
6) The query is tokenized
7) Tokenized query is encoded into vectors
8) Vectors are searched in the index
9) Nearest neighbor results are returned

Figure 38. A conceptual architecture for end-to-end search using transformer encoded vectors (Grainger et al., 2021)

In this research, the applied Machine Learning approach delivered by the MLOps cycle is a semantic search solution utilizing Language Model categorized in Natural Language Processing tasks.

The data preparation step is accomplished by choosing a dataset followed by data understanding, cleaning, and transformation. The current research is mainly concentrated on the Develop, Train, and Review steps in the MLOps cycle as they are the core of the solution's functionality. With the assumption that the proper data is prepared and ingested into the pipeline, the solution will be developed by selecting the suitable model for the planned task and then evaluating the performance.

In this project, the solution is developed with two different models of Transformers with various number of parameters, different capacities in input acceptance, and the size of generated embedding vector. Regarding these criteria, the models can be compared in being aligned with human judgment alongside resource consumption and language understanding. The chosen models include *DistilBERT* from the sentence-transformers library with 66M (66 million) parameters mapping the input text into a 768-dimensional dense vector, and *GPT-2*$_{large}$ from the *Transformers* library with 774M parameters converting the input text into a 1280-dimensional dense vector. The *DitilBERT* takes input consisting of a sequence of 512 tokens but *GPT-2*$_{large}$ can accept an input of 1024 tokens. The differences show that *GPT-2*$_{large}$ is a larger model considering the more oversized number of the parameters and greater input capacity provisioned to capture more context.

Therefore, the naive hypothesis of the current research is formed around that. In Table 3 the parameter size of some pre-trained models available on Hugging Face are summarized.

Model	Parameter size
DistilBERT	66M
GPT-2	1.5B
GPT-2_{large}	774M
DistilGPT-2	82M
GPT-2_{medium}	355M
GPT-j-6b	6B
BERT-uncased-large	340M
BERT-uncased	110M
BART-large	140M
DeBERTa-v2-xlarge-mnli	900M
RoBERTa-large	354M

Table 3. Parameter size of some language models on Hugging Face including the selected models in the current project.

When the model is chosen, the development will continue with capturing the embeddings and making an index of them to search through. Then it will be followed up with feeding the test queries to the system and producing the candidates which can be re-ranked as well.

The development process will be completed by assessing the model performance for the provided data. Here the judgment list besides an appropriate evaluation metric plays a crucial role in achieving the satisfactory level of generated output. Sometimes it is essential to inject the domain-specific wisdom into the engine by re-training it with the prepared data (fine-tuning) and then evaluating again till reaching the level of satisfaction. It can be repeated until coming up with the best setting of model hyperparameters in re-training and it would be the last step before reviewing all requirements to deploy the model into production. After deployment, the solution will encounter the actual users, which is referred to as Inference. The interaction with the user will be monitored regarding the latency, and user satisfaction with the results and necessary changes will be made in the cycles by analyzing the performance. The user interaction and feedback could be the augmentation data utilized for re-ranking which is another interesting topic in boosting the search results but is beyond the scope of this research.

The semantic search pipeline is implemented utilizing Python programming language, Pandas as the software library for Python to manipulate and analyze data, spaCy an open-source software library for natural language processing in Python, Hugging Face a platform to build, train and deploy state of the art models powered by the reference open-source in machine learning, Sentence-Transformers a Python framework for state-of-the-art sentence, text, and image embeddings, Transformers a State-of-the-art Machine Learning library for PyTorch and TensorFlow, which provides APIs and tools to easily download and train state-of-the-art pre-trained models, Jina AI[2] a state-of-the-art LMOps, MLOps and cloud-native technologies, Docker Containers as a delivery mechanism, Docker is a platform for running virtual machine images with all operating system configuration and dependencies needed, and Kaggle a data science competition platform and online community of data scientists and machine learning practitioners under Google LLC which enables users to find and publish datasets, explore, and build models in a web-based data science environment. The codes are executed on Google Colab notebooks using Google Cloud servers including TPU, and GPU. The Python codes are available on GitHub[3]. It should be mentioned that text, as one type of unstructured data, is the primary modality of data in this research.

The primary dataset employed as the collection of contents includes several fields of information where the principal field is the raw text of various lengths from multiple words to long passages. This is a challenge that could be overcome by splitting the passages into chunks of the maximum length of input that the language model can accept or to keep it simpler by just truncating the input to the model to the maximum length. It must be noticed that the latter solution may result in forfeiting some parts of the information.

The dataset includes 34886 documents processed with the language model to produce the required knowledge for searching semantically through it. Hence, it takes a prolonged time to be processed and consumes considerable computing resources. To accomplish the process with a free account

[2] Jina AI - Your Portal to Multimodal AI

[3] https://github.com/eliehv/Semantic_Search_using_LLM
https://github.com/eliehv/Knowledge-based-semantic-search

on Google Colab, the data is split into smaller groups and fed into the language model in batches. Ultimately, the captured knowledge compacted into embeddings is concatenated into a single data frame.

Among introduced metrics in Section 3.2, recall, precision, and MRR are calculated, and the results are analyzed in Chapter 4. The precision and recall are selected since they can smoothly explain the outcomes and complement each other's insights, but they don't consider the rank of the items in the candidate list in the calculation so, to complete the simplicity and explain-ability with the weighted rank the MRR is added to the selected metrics.

4.2 Generate Embedding using Language Models

According to the previous chapter discussion, two language models with different properties are selected for the experimental setting regarding essential comparison characteristics including the input capacity, parameters quantity, and embedding vector dimension. The contextual and relational knowledge will be captured from the movie plots (*Plot* column in the dataset) using the selected language model by generating embeddings which are the dense vector representation of the captured knowledge. The code snippet attached in Appendix D demonstrates the required steps.

One of the employed models is "*msmarco-distilbert-dot-v5*"[4] from the *sentence-transformer* models accessible on the *Hugging Face* platform. *Sentence-transformers* is a Python framework for state-of-the-art sentence, text, and image embeddings. The initial work is described in (Reimers & Gurevych, 2019). This framework is based on PyTorch and Transformers including a large collection of pre-trained models tuned for various tasks. It can be used to compute sentence or text embeddings and then compare them with cosine similarity to find sentences with a similar meaning which can be useful for semantic textual similarity, semantic search, or paraphrase mining (*SentenceTransformers Documentation — Sentence-Transformers Documentation*, n.d.). "*msmarco-distilbert-dot-v5*" model has been trained on 500K of query-answer pairs from the *MS MARCO* dataset which is introduced in Appendix A.

"It is critical that you choose the right model for your type of task. It is mostly distinguished by the type of data it has been trained on. Also, models tuned for cosine-similarity will prefer the retrieval of short documents, while models tuned for dot-product will prefer the retrieval of longer

[4] sentence-transformers/msmarco-distilbert-cos-v5 · Hugging Face

documents." (Verma, 2021). According to the importance of the similarity function corresponding to the task which contains long texts in this project, dot-product is used as a similarity function in Bi-Encoder for the "*msmarco-distilbert-dot-v5*" model.

A pre-trained model only performs properly if an input that is tokenized with the same rules that are used to tokenize its training data is fed into it. Correspondingly, when the model and corresponding tokenizer are obtained, the input text will be passed to the tokenizer and then to the model to compute the corresponding embeddings. This process is defined as the *get_embedding()* function in the code. The free accessible memory on Google Colab cannot handle all Plots of the dataset at once so, it is divided into groups based on the Genre, and the get_embedding() function is applied to all groups independently and ultimately all groups concatenated.

The produced embedding vectors besides the number of tokens in each input text (movie plot) will be added to the data frame for further utilization.

The same process is executed with the other selected model, "*gpt2-large*"[5]. This model is accessible via the Transformers library which provides APIs and tools to easily download and train state-of-the-art pre-trained models. The "*gpt2-large*" model is described in detail in Section 2.3.5.2. It is mentioned, on the OpenAI web page[6], that OpenAI embeddings are normalized to length 1 which means that the cosine similarity can be computed slightly faster using just a dot product and it results in the identical ranking. Therefore, in the Bi-Encoder procedure to find the nearest neighbors, cosine similarity will be used for the embeddings generated by "*gpt2-large*".

The embedding generating process is completed in a hugely different span of time which is summarized in Table 4. The smaller model, which is a distilled version of BERT and task-oriented for sentence similarity, takes a much smaller span of time to be accomplished.

Language Model	**Embedding Generation Time**	**Processor Type**
gpt2-large	88 hours	Google Colab TPU
msmarco-distilbert-dot-v5	6 hours	Google Colab TPU

Table 4. Processing time for embedding generation

[5] gpt2-large · Hugging Face

[6] Embeddings - Frequently Asked Questions | OpenAI Help Center

4.3 Results and Analysis

4.3.1 Bi-Encoder

After producing embedding for all Plots and storing them, the queries can be fed to the system. Then, the queries' embeddings will be generated using the same model. In the next step, the results will be returned based on the similarity between the embeddings. These steps happen in the architecture shown in Figure 38 and the code provided in Appendix C. Table 5 demonstrates the primary results, to answer query number 1 in the provided test query set created by Bi-Encoder for both models. This table shows only the 5 top nearest neighbors (movies) to the query even though the number of the top *k* results is set to 50 in this experiment.

Query	Model	Positional rank	Title of the movie
Query 1: "Artificial intelligence based action movie"	*msmarco-distilbert-dot-v5*	1	**D.A.R.Y.L.**
		2	**Ra.One**
		3	Chanakya Chandragupta
		4	The Last Starfighter
		5	**Chappie**
	gpt2-large	1	Walk Like a Dragon
		2	Murder in the Air
		3	Sanyasi Mera Naam
		4	**Crosstalk**
		5	Rokto

Table 5. The results of Bi-Encoder for both selected models

It is evident, from the information provided in Table 5, that the "*msmarco-distilbert-dot-v5*" outperformed *"gpt2-large"* which means it provided more relevant candidates for example

"D.A.R.Y.L", "Ra.One", and "Chappie". In addition, if further results from the model "*msmarco-distilbert-dot-v5*" are scanned, it will be displayed that the movie *"Transcendence"* which seems to be a relevant item to the query, occurs in the 15th rank out of 50 returned items. It could be concluded that re-ranking may direct to a better performance regarding the more relevant items appearing in the higher positions.

These outcomes lead to a premature conclusion that *"gpt2-large"* may not compete with the small task-oriented pre-trained model in recognizing the nearest neighbors even though it is a much larger language model with a more extensive contextual capacity.

The outcomes of the current research at this stage are compared with the results of a semantic search engine implemented using the "*msmarco-distilbert-base-dot-prod-v3*" by Verma 2021 which is shown in Table 6 (Verma, 2021). That search engine takes the same dataset employed in this experimental research. There is a common movie in the 5 top candidates returned by using "*msmarco-distilbert-base-dot-prod-v3*" and "*msmarco-distilbert-dot-v5*" while, apparently, there is no intersection with *"gpt2-large"*. Version 5, *v5*, of the *"msmarco-distilbert-dot",* contains the *v3* cosine-similarity models but with an additional normalized layer on top besides being tuned for dot-product which makes it suitable to accept longer passages (*MSMARCO Models — Sentence-Transformers Documentation*, n.d.).

Query	**Model**	**Positional rank**	**Title of the movie**
Query 1: "Artificial intelligence based action movie"	*msmarco-distilbert-base-dot-prod-v3*	1	The Cape Canaveral Monster
		2	**Small Soldiers**
		3	**Chappie**
		4	Armed Response
		5	Galactic Armored Fleet Majestic Prince: Genetic Awakening

Table 6. Bi-Encoder results using "msmarco-distilbert-base-dot-prod-v3" (Verma, 2021)

4.3.2 Re-ranking

According to the what is explained in Chapter 3 regarding the similarity score and re-ranking methods, in this experimental research, Cross-Encoder and BERTScore are utilized to calculate the similarity between the queries and the candidate's Plots in order to re-order the rank of the candidates to achieve the more reasonable recommended list.

The Cross-Encoder resembles the similarity between the pairs of queries and the Plots of the items in the candidate list for example query 1 and the plots of the movies mentioned in Table 5. According to Figure 31, the pairs will be passed to a language model to be compared by their contextual embeddings and classified as similar or dissimilar. The Cross-Encoder calculates the score of the similarity that can be used for classifying as similar or dissimilar. The language model utilized in the Cross-Encoder is *"cross-encoder/mmarco-mMiniLMv2-L12-H384-v1"*[7] which is a version of *MiniLMv2* described in Section 2.3.5.4 trained on the *MS MARCO* dataset. The re-ranking code using Cross-Encoder is provided in Appendix E.

Another model that is used as a re-ranker is BERTSCore which will compare the pairs of queries and the Plots token-wisely as demonstrated in Figure 33. The BERTScore uses a language model as the foundation of the comparison which is *"deberta-xlarge-mnli"*[8] a version of the DeBERTa and it is fine-tuned for the Natural Language Understanding task. The re-ranking code using BERTScore is attached in Appendix F.

It should be mentioned that in the re-ranking stage, the Plots' text is split into smaller chunks. The code for splitting Plots into chunks is shown in Appendix G. The chunk size is set to 50 tokens. This number is selected after an empirical examination of various values indicating that the smaller quantity of tokens in the input sequence operates better than more tokens. This indication needs to be investigated more in separate research because it contradicts the fact that the language models used in Cross-Encoder and BERTScore are large models able to accept a long sequence of tokens as input. A reason could be the token-wise comparison in BERTScore that may reduce the input capacity of BERTScore compared to the *"deberta-xlarge-mnli"* by itself. On the other hand, computing the similarity on chunks of Plots equips Method 2, one of the proposed methods in creating a judgment list, with the requirements.

The chunk comparison provides two possible approaches for deciding on the better matches, which is introduced in Section 3.3, also it will assure that each chunk will fit into the language model. At the same time, it may compromise capturing the whole context at once which is one of the prominent attributes of the Large Language Models (LLM).

---

[7] cross-encoder/mmarco-mMiniLMv2-L12-H384-v1 · Hugging Face
[8] microsoft/deberta-v2-xlarge-mnli · Hugging Face

Table 7 outlines the re-ranked list of items by applying the Cross-Encoder re-ranker on the candidate list provided by "*msmarco-distilbert-dot-v5*" and Table 8 abstracts the re-ranked list by employing the BERTScore.

Cross-Encoder Re-rank position	**Bi-Encoder position**	**Title**
1	1	**D.A.R.Y.L.**
2	7	The Legend of Wisely
3	2	**Ra.One**
4	14	Universal Soldier: The Return
5	38	**Evolver**
6	15	**Transcendence**
7	11	Leonard Part 6
8	43	**Terminator 2: Judgment Day**
9	6	Black Eagle
10	9	You Don't Mess with the Zohan

Table 7. Re-ranking with Cross-Encoder using "msmarco-distilbert-dot-v5"

As it is reported in Table 7, the Cross-Encoder ranked the more relevant items in higher positions as it is expected. For example, *"Transcendence"* is driven from the 15th to the 6th rank, and other relevant items like "Universal Soldier: The Return" and "Terminator 2: Judgment Day" appeared in higher positions even though an irrelevant item "The Legend of Wisely" is appeared in the high rank which is not valid.

To obtain another perspective of the similarity between queries and the chunks in the Plots, a similarity matrix which calculates the token-wise similarity score by BERTScore model will be practical. The top 1 re-ranked item with Cross-Encoder is *"D.A.R.Y.L."* baked on the Table 7 information and its highest scored chunk is *"Daryl acronym data analyzing robot youth lifeform* *barret oliver experiment artificial intelligence create government"*. Figure 39 shows the heatmap graph of the similarity matrix for that chunk and query 1. The similarity matrix shows that the context and keywords in the selected chunk perfectly match the query intent which is desired.

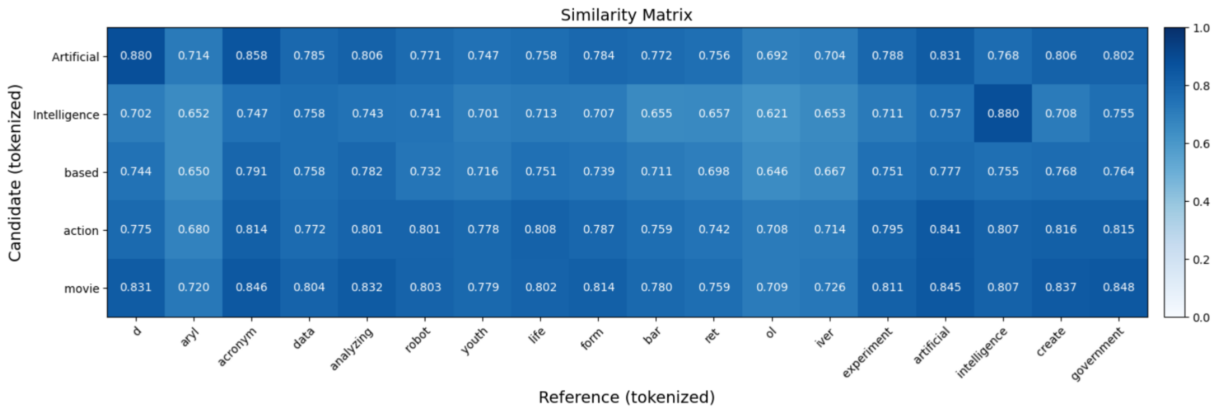


Figure 39. Similarity matrix of the highest-scored chunk in the top 1 item re-ranked by Cross-Encoder and query 1

The 2nd top re-ranked item with Cross-Encoder is *"The Legend of Wisely"* movie and its highest-scored chunk is *"Samuel Hui play wisely big budget hong kong movie production unit film scene great pyramids scene Nepal car chase crash chase horseman plenty fight way"* which seems to be an irrelevant candidate, but it attained a high rank which is unwelcome. So, the heatmap similarity matrix for that chunk and query 1 is illustrated in Figure 40 which may provide a more explicit vision of the recognized similarities by the model. Although it is not explainable accurately, it can be reviewed in Figure 40 that the words *movie*, *film*, and *scene* in the selected chunks match the word *movie* in the query with a high score and the word *wisely* may provide some contextual similarity. One unexpected observation in this matrix is that the word *Samuel*, which is a person's name, is tokenized in two sub-words, and the first part, *Sam*, obtained the highest similarity score with the word *artificial*. This indicates that the tokenizer functionality could affect the outcomes dramatically.

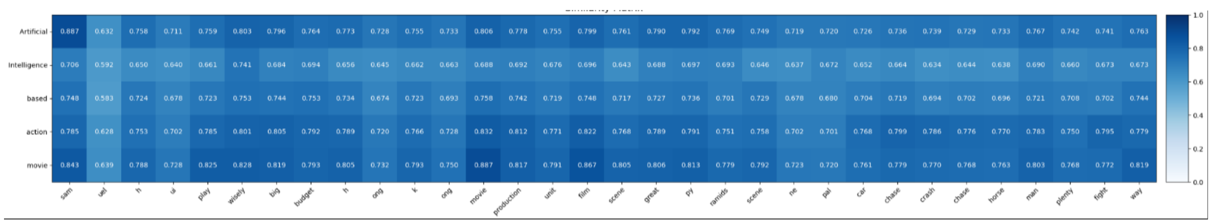
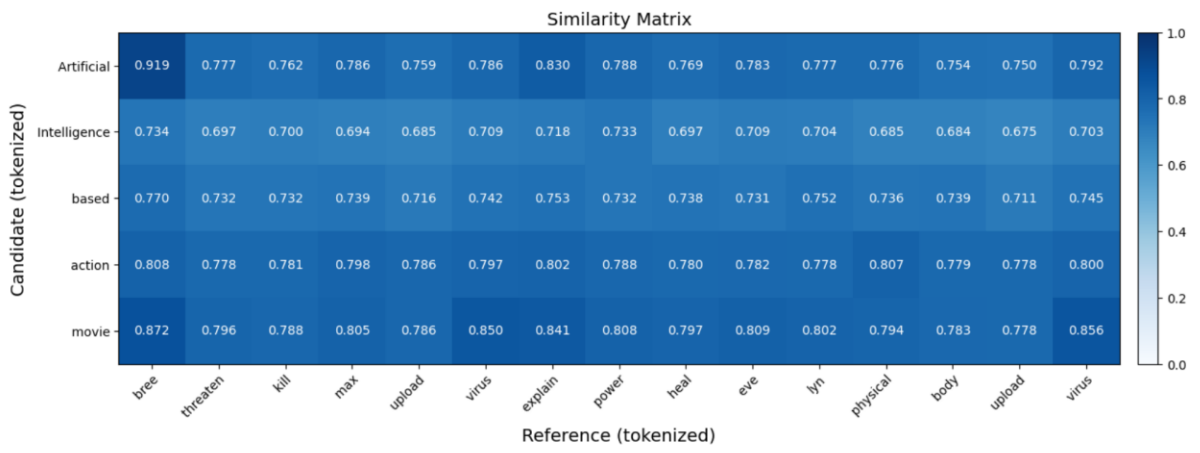


Figure 40. Similarity matrix for the tokens of the highest scored chunk in the 2nd top item re-ranked by Cross-Encoder and query 1

Likewise for the 6th top item in the re-ranked list created by Cross-Encoder, *“Transcendence”*, the highest scored chunk is *“bree threaten kill max upload virus explain power heal evelyn physical body upload virus”*. Figure 41 shows its similarity matrix. In this example also the person’s name, *Evelyn*, is tokenized in two separate parts which is not desired. This could signify that the Named Entity Recognition process would be effective if it is added to the preprocessing pipeline. This idea can be investigated in a further research project.

By observing the similarity matrices, the trade-off between the input capacity of the language model used by BERTScore and the capacity of token-wise comparison will be more obvious which emphasis on the importance of exploring its ambiguity more precisely. It could be inferred that more additional context would result in better decisions made by the Language Model in addition to more explain-ability.

The results in Table 8, report that BERTScore functioned more ambiguously regarding moving the more relevant candidates to the higher ranks. For instance, *“Transcendence”* as a relevant item occurred in rank 1 which is perfect but *“D.A.R.Y.L.”* as a matched item is driven down to the list and more irrelevant items like “Kantri” appeared in the 10 top ranks which are not desired.



	bree	threaten	kill	max	upload	virus	explain	power	heal	eve	lyn	physical	body	upload	virus
Artificial	0.919	0.777	0.762	0.786	0.759	0.786	0.830	0.788	0.769	0.783	0.777	0.776	0.754	0.750	0.792
Intelligence	0.734	0.697	0.700	0.694	0.685	0.709	0.718	0.733	0.697	0.709	0.704	0.685	0.684	0.675	0.703
based	0.770	0.732	0.732	0.739	0.716	0.742	0.753	0.732	0.738	0.731	0.752	0.736	0.739	0.711	0.745
action	0.808	0.778	0.781	0.798	0.786	0.797	0.802	0.788	0.780	0.782	0.778	0.807	0.779	0.778	0.800
movie	0.872	0.796	0.788	0.805	0.786	0.850	0.841	0.808	0.797	0.809	0.802	0.794	0.783	0.778	0.856

Figure 41. Similarity matrix for the tokens of the highest scored chunk in the 6th top item re-ranked by Cross-Encoder and query 1.

The highest-scored chunk in the top 1 re-ranked item with BERTScore, *“transcendence”*, is *“dr. caster johnny depp scientist research nature sapience include artificial intelligence”*. The corresponding similarity matrix is depicted in Figure 42, notice that this is a highly relevant item.

BERTSore re-rank position	**Bi-Encoder position**	**Title**
1	15	**Transcendence**
2	25	**Nemesis**
3	27	Kantri
4	1	**D.A.R.Y.L.**
5	5	**Chappie**
6	38	**Evolver**
7	33	**Terminator Salvation**
8	49	Rangamati
9	41	Katha
10	32	A Gentleman

Table 8. Re-ranking with BERTScore using "msmarco-distilbert-dot-v5"

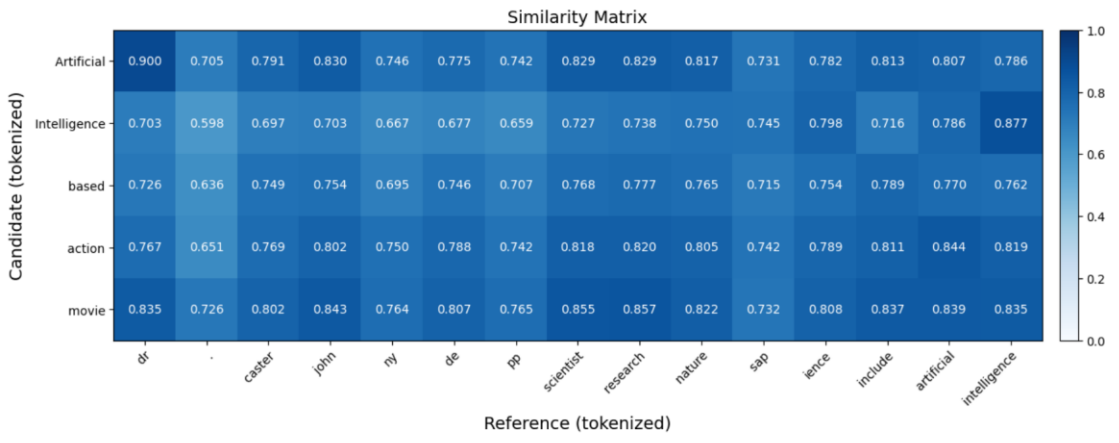


Figure 42. Similarity matrix for the tokens of the highest-scored chunk in the top 1 item re-ranked by BERTScore and query 1

It should be commented that the highest-scored chunk for the same movie, *"Transcendence"*, is different in Cross-Encoder and BERTScore which can be reviewed in Figures 41 and 42. This implies that token-wise similarity measurement which happens in BERTScore operates differently from finding the nearest neighbor in embedding vector space which emerges in Cross-encoder. Also, it should be noticed that distinct language models are used. If Figures 41 and 42 are studied simultaneously, it will be understood that the selected chunk by BERTScore contains more tokens with high similarity scores for instance "*dr*", "*scientist*", "*research*", "*artificial*", and "*intelligence*". However, the person's name "*John*" gained a high score as well which there is no explicit explanation for it. It should be considered here that BERTScore is cited as an outstanding method in approximating human judgment in text similarity tasks. So, it could be concluded that it is performed properly as a re-ranker although it ranked some irrelevant items in the top 10 which is not explainable.

The 3rd top re-ranked item with BERTScore is "*Kantri*" which is an irrelevant item. Its highest scored chunk is *"movie stylish fight good action sequence"*. This movie's selected chunk is shown in Figure 43 on the similarity matrix of the tokens. So, it can be examined more accurately with an explainability objective. There is a high score match between the words "*artificial*" and "*movie*" which is observed in other matrices also. It seems that the chunk is selected mostly based on word meaning similarity than contextual similarity which could be remarked as another evidence to explore an extended chunk size in future research.

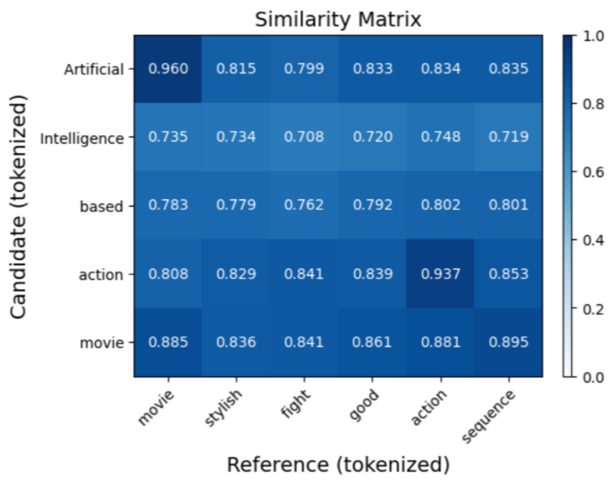


Figure 43. Similarity matrix for the tokens of the highest-scored chunk in the 3rd top item re-ranked by BERTScore and query 1

The re-ranking process is executed also for the top 50 candidates created in Bi-Encoder using *"gpt2-large"*. Both types of re-ranker are employed in this case as well. Tables 9 and 10 report the results for Cross-Encoder and BERTScore re-ranking respectively.

Cross-Encoder re-rank position	**Bi-Encoder position**	**Title**
1	17	Television Spy
2	14	The Man Who Made Diamonds
3	25	White
4	9	The Green Pastures
5	40	Bullets and Saddles
6	45	C.H.O.M.P.S.
7	32	Nothing Else Matters
8	43	Wide Boy
9	29	Oyee
10	20	Key to Harmony

Table 9. Re-ranking with Cross-Encoder for candidates created by gpt2-large.

After Re-rank with Cross-Ecoder, the best item in the candidate list returned by Bi-Encoder using *"gpt2-large",* which is "Crosstalk", appeared in the 34th position. It could mean that re-ranking operated reversely in this case. In other words, the most relevant item recognized by *"gpt2-large"* does not appear in the top 10 after re-ranking which is undesirable. But it appeared in the 22nd rank by BERTScore re-ranking which seems a bit better although it is not still in the top 10 list and no relevant item occurred in the top 10.

A more accurate review of "*msmarco-distilbert-dot-v5"* scores created with both re-rankers indicates that those scores are mostly concentrated in the middle of the score range for BERTScore and spread in a shorter range for Cross-Encoder which could be considered as a booster to the indication that this model performs better than *"gpt2-large"* in this downstream task. In other words, "*msmarco-distilbert-dot-v5"* seems to be more consistent. However, the range of scores and the variations that happens in re-ranking is more considerable for the *"gpt2-large"*. The score variation is even more expansive for BERTScore.

BERTScore re-rank position	**Bi-Encoder position**	**Title**
1	11	Ghamandee
2	1	Walk Like a Dragon
3	8	Aadesh - Power of Law
4	16	Hum Farishte Nahin
5	36	Kote
6	10	Jawaani
7	29	Oyee
8	7	Seesa
9	32	Nothing Else Matters
10	49	Wind

Table 10. Re-ranking with BERTScore for candidates created by gpt2-large.

A more accurate review of the results created by *"gpt2-large"* reveals a prominent insight which shows that candidates suggested by *"gpt2-large"* are biased on the Drama and Romance genres, for example for query 1 the candidate list contains fewer items from the Science Fiction genre while it is supposed to be the main genre correspondent to this query. The candidate lists provided by both employed language models and for both examined re-rankers are reported in Appendices K and L. By studying the tables in those Appendices, it would be revealed that *"gpt2-large"* provide a more general view of the captured intention from the queries and the candidates. In addition, it is observable that it provides a wide range of genres in the suggested items. For example, "*msmarco-distilbert-dot-v5*" results in candidates mostly from the Science Fiction genre for queries 1 and 2 while *"gpt2-large"* candidates include more variety of genres. This behavior could be derived from being a general-purpose model, not tuned for text similarity, and capturing the context in a general view to predict the next word in the sequence from the only left side context to generate text. This underlines that this model may not outperform regarding the specific task to find similarities, but it can be generalized well which can be considered as a complementary attribute. This attribute is utilized in the proposed methods for creating a judgment list to provide an automatic performance evaluation pipeline. It should be accentuated that *"gpt2-large"* brought up a candidate "Crosstalk" which is not recognized as a relevant item by the other language model. However, the results generated by "*msmarco-distilbert-dot-v5*" contains mostly Science Fiction and Adventure genre. The results of *"gpt2-large"* seems a bit better for queries 2 and 3. On the other hand, it should be considered that there are some items among the candidates produced by *"gpt2-large"* for instance "Crosstalk" for query 1 which is a relevant match, but it appears in neither Cross-Encoder nor Bert-Score top 10 results. It could be indicated that re-ranking is not performing well for *"gpt2-large"* candidates.

The fact that *"gpt2-large"* can provide a few relevant items, that are not suggested by the other model at all, could be helpful for better generalization. Its poor performance despite its huge contextual capturing capacity, could be explained by not being conditioned for a specific task in the embedding generation phase. More precise research on the conditions and experimental settings, e.g., summarizing or splitting the text before being fed into the model to generate embedding, can be performed to reach a more accurate conclusion but in general, according to the reported results in this experiment, it seems that the model is not appropriate for embedding similarity-based tasks like information retrieval and semantic search.

The data and setting of the experiments can be adjusted more precisely based on the finding of this research to explore more potential discoveries which is not able to be done in the current research due to the lack of time and resources (lack of resources makes the computations take more time). For example, the input sequence of tokens besides the number of tokens considered in splitting the Plot into chunks in re-ranking can be explored more. It seems that the small number of tokens in the chunks makes the re-ranking process function more like a synonym or keyword match because it contains less context. This indication can be investigated in further research as well.

The different language models with various architectures and parameter sizes make the re-ranking process take distinct time spans to be accomplished. The recorded computational time spans for the Cross-Encoder and BERTScore are documented in Table 11. The language model used in the experimental setting of the BERTScore re-ranking is *"microsoft/deberta-xlarge-mnli"*. Based on the provided data, BERTScore takes much more time to compute the scores. The size and attribute of the employed language model in re-ranking directly affect the consumed amount of time.

Re-ranker	**Time (minutes)**	**Number of candidates**	**Language Models**
CrossEncoder	2	50	"cross-encoder/mmarco-mMiniLMv2-L12-H384-v1"
BertScore	34	50	"microsoft/deberta-xlarge-mnli"
BertScore	12	50	default

Table 11. Re-ranking time spans

Bi-Encoder is a perfect method to find the semantically similar items to the input query in a huge collection of documents because it just computes the similarity between already stored embedding vectors so, it can produce the results fast enough for the inference time. On the other hand, re-rankers like Cross-Encoder can provide more satisfying results but it takes dramatically longer time since it computes the similarity of each pair of query and text. Therefore, their power will be unlocked as a re-ranker to be applied to a smaller collection of documents generated by a simple Bi-encoder. This a trick to achieve the preferably ranked results in a shorter time. In addition, as it is discussed in Sections 3.3.3 and 4.3.3, they can be utilized in creating the requirements to implement an automatic evaluation pipeline for information retrieval systems.

4.3.3 Generating Judgment List

In this section the results of the various labeling methods discussed in Section 3.3.3 are documented and analyzed. These results encompass the evaluation metrics calculated for each of the introduced labeling methods.

As mentioned before, in the present research, recall, precision, and MRR are utilized to evaluate the performance of language models in the semantic search downstream task.
Recall is informative because it deals with *FalseNegatives where* fewer *FalseNegatives* mean that in the suggested list of items, there are more relevant items. If the recall is equal to 1 it points that there is no relevant item that is not included in the suggested items.
Precision presents a useful insight into the performance as well since the closer its value is to 1, implies that there are fewer *FalsePositives* in the offered list of relevant items which means including fewer irrelevant items. Although it should be noticed that *FalsePositives* could enhance the chance of including more variety of contextual concepts. Therefore, it seems that the precision value could be more passable.

4.3.3.1 Method 1

This method as demonstrated in Figure 35, utilizes both re-ranking methods to assign a similarity score to each query-chunk pair for all queries and their candidates created by Bi-encoder. The re-ranked lists based on the highest-scored chunks are made by both re-rankers. Then, the intersection of them is computed to provide a single set of relevant items as an approximation of human judgment. Bi-encoder provides the top 50 candidates using the *"msmarco-distilbert-dot-v5"* language model. So, the number of the re-ranked items that can be selected to join in the intersection is the parameter of this method. In the experimental setting for Method 1, the values of 5 and 30 are chosen to be investigated for this parameter. The code for this labeling method is attached in Appendix H. Table 12 reports the evaluation metrics by applying the judgment list created with Method 1 and Table 13 shows the metrics when *"gpt2-large"* is used as the language model in Bi-Encoder. Recall, precision, and MRR are calculated for the candidate list and re-ranked outputs by both re-rankers.
It should be noted that precision and recall are calculated for each query separately, but MRR is calculated over all involved queries. So, in each row of the tables, there is one single value for MRR computed over 6 queries while there are 6 values (one value per query) for precision and recall that form a list.

Method's parameter	**Re-ranking method**	**MRR@6**	**Precision**	**Recall**
5	Bi-Encoder	0.3192	[0.02, 0.02, 0.04, 0.08, 0.08, 0.08]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
5	Cross-Encoder	0.7917	[0.02, 0.02, 0.04, 0.08, 0.08, 0.08]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
5	BERTScore	0.75	[0.0204, 0.02, 0.0408, 0.0816, 0.08333, 0.08]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
30	Bi-Encoder	0.6528	[0.46, 0.36, 0.36, 0.36, 0.42, 0.42]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
30	Cross-Encoder	0.8333	[0.46, 0.36, 0.36, 0.36, 0.42, 0.42]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
30	Bi-Encoder	0.6722	[0.4694, 0.36, 0.3674, 0.3674, 0.4375, 0.42]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Table 12. Performance evaluation metrics calculated by applying Method 1 with "msmarco-distilbert-dot-v5".

One dominant observation is that the recall is all 1 for every setting. It means that there is no *FalseNegative*. It happened because it is assumed that all potential matches are returned by Bi-Encoder. In other words, it is supposed that Bi-Encoder functioned perfectly, and the result candidates include all possible options which is not true. It should be considered also that only the 50 top results of the Bi-Encoder are chosen to be passed to the re-ranker.

According to the reported data in Tables 12 and 13, precision values have changed by different values set for the parameter also by the employed language model, but they are almost fixed per language model and determined parameter. The fixed precision values in the same setting for the language model and parameter value points that the number of *FalsePositives* are stable under identical conditions which means is apparent because no irrelevant document will be added to the candidates' list in that situation.

Method's parameter	**Re-ranking method**	**MRR@6**	**Precision**	**Recall**
5	Bi-Encoder	0.2355	[0.08, 0.08, 0.02, 0.08, 0.08, 0.08]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
5	Cross-Encoder	1.0	[0.08, 0.08, 0.02, 0.08, 0.08, 0.08]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
5	BERTScore	0.88	[0.08, 0.0833, 0.0208, 0.0851, 0.0833, 0.0816]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
30	Bi-Encoder	0.3333	[0.28, 0.38, 0.48, 0.36, 0.34, 0.32]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
30	Cross-Encoder	0.6806	[0.28, 0.38, 0.48, 0.36, 0.34, 0.32]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
30	BERTScore	0.5988	[0.28, 0.3958, 0.5, 0.3830, 0.3542, 0.3265]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Table 13. Performance evaluation metrics calculated by applying Method 1 with "gpt2-large"

The best precision happened when the parameter is set to 30 and "*msmarco-distilbert-dot-v5*" is employed to generate embeddings. This shows that this is a proper setting for taking advantage of this method and it confirms the previous conclusion that "*msmarco-distilbert-dot-v5*" outperform *"gpt2-large"*.

The values in the precision list do not vary in a wide range. However, it could be observed that the precision is better for query 1 in the best row which is not the case in other rows.

It is obvious from both tables that the greater value of the parameter, 30, leads to better performance for both language models which means including more items increases the chance of appearing more relevant items in results, *TruePositives*. Both tables equally contribute to concluding that the Cross-Encoder re-ranker surpasses the BERTScore.

4.3.3.2 Method 2

This method considers each chunk of a plot as a separate item in the re-ranking process. Then the chunks that belonged to the same movie and appear more frequently as high-scored chunks determine the relevant items. For example, if the top 100 best-scored chunks are selected, the most frequent movies among selected chunks will be chosen as relevant documents. So, the top highest-scored chunks are grouped by movie, and the frequency of each movie appearance is counted then the groups that have at least k (*min_num_group_items*) members will be selected. This process repeats for both Cross-Encoder and BERTScore then the intersection will be returned as relevant items. The Python code that implemented Method 2 is attached in Appendix I and Table 14 summarizes the results made using the "*msmarco-distilbert-dot-v5"* language model and Table 15 documents the results where the *"gpt2-large"* is applied.

min_num_group_items	**Re-ranking method**	**MRR@6**	**Precision**	**Recall**
5	Bi-Encoder	0.5516	[0.54, 0.46, 0.18, 0.38, 0.34, 0.28]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
5	Cross-Encoder	0.8333	[0.54, 0.46, 0.18, 0.38, 0.34, 0.28]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
5	BERTScore	0.5611	[0.5510, 0.46, 0.1837, 0.38778, 0.3542, 0.28]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
10	Bi-Encoder	0.4750	[0.34, 0.28, 0.04, 0.08, 0.1, 0.04]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
10	Cross-Encoder	0.5243	[0.34, 0.28, 0.04, 0.08, 0.1, 0.04]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
10	Bi-Encoder	0.3854	[0.3469, 0.28, 0.0408, 0.0816, 0.1042, 0.04]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Table 14. performance evaluation metrics calculated by applying Method 2 with "msmarco-distilbert-dot-v5"

The minimum number of members in each group, *k*, is the parameter of Method 2. Two value is considered as the empirical setting for this method including values equal to 5 and 10.

The number of chunks included in each group is not large when *"gpt2-large"* is employed. The maximum number of chunks in a group is 9. Also, there are plenty of groups with only one member. Therefore, instead of considering a fixed value for the parameter of Method 2, a variable value formulates as *max(number of items in groups) - k* is recommended in this situation.

Plenty of groups with a single member indicate that *"gpt2-large"* capture totally different contextual information in its embeddings, maybe more generalized information. This observation confirms the indications made with Method 1 regarding the difference that is observable in the *"gpt2-large" operation* including more variety of genres and tends to retrieve the *Drama* and *Romance* genres which are the most portion of the dataset.

According to the reported data in Table 14, a smaller value for parameter *k* results in better performance which is determined by a higher value of MRR. For example, the value of 5 means that the groups with more than 5 members can be included in the selection so, there is more chance for relevant items to be considered in the judgment list. This conclusion is verified in Table 15 as well, in this case, *max(number of items in groups) - 9* points that any groups that have at least one member can be included which obviously resulted in an MRR of 1. On the other hand, it should be considered as a trade-off in this method. Regarding that *max(number of items in groups) - 5* would be considered as the baseline.

MRR is the main metric to evaluate the performance of the finding of re-ranked nearest neighbors since it considers the rank of items in the calculation process. As one of the objectives is to determine the better re-ranker it is a perfect choice for evaluation.

It is approved in both tables that Cross-Encoder transcends the BERTScore regarding re-ranking.

It should be noted that in the process of creating the judgment list with Method 2 using *"gpt2-large"*, where the intermediate results are checked, interesting outcomes are observed. The re-ranked list based on the frequency of the chunks for both re-rankers contain plenty of common items. In other words, there are many common items in the re-ranked lists as they could be considered identical. It could signify that the contextual features captured in the embedding provided using *"gpt2-large"* are scored similarly with both BERTScore and Cross-Encoder. It points outs the difference between the general-purpose pre-trained model and the task-oriented tuned model. It could be another verification of the consistency of the features and contextual content captured by *"gpt2-large"* even though it does not perform appropriately for semantic text similarity purposes according to the results reported in Sections 4.3.1 and 4.3.2.

min_num_group_items	Re-ranking method	MRR@6	Precision	Recall
max(number of items in groups) - 1	Bi-Encoder	0.02	[0.02, 0.02, 0.02, 0.02, 0.08, 0.02]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
max(number of items in groups) - 1	Cross-Encoder	0.36	[0.02, 0.02, 0.02, 0.02, 0.08, 0.02]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
max(number of items in groups) - 1	BERTScore	0.07	[0.02, 0.02, 0.02, 0.02, 0.08, 0.02]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
max(number of items in groups) - 5	Bi-Encoder	0.53	[1.0, 0.04, 0.4, 0.94, 0.96, 0.1]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
max(number of items in groups) - 5	Cross-Encoder	0.92	[1.0, 0.04, 0.4, 0.94, 0.96, 0.1]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
max(number of items in groups) - 5	BERTScore	0.60	[1.0, 0.04, 0.42, 1.00, 1.00, 0.10]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
max(number of items in groups) - 9	Bi-Encoder	1.00	[1.0, 0.96, 0.96, 0.94, 0.96, 0.98]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
max(number of items in groups) - 9	Cross-Encoder	1.00	[1.0, 0.96, 0.96, 0.94, 0.96, 0.98]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
max(number of items in groups) - 9	BERTScore	1.00	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Table 15. performance evaluation metrics calculated by applying Method 2 with “gpt2-large”.

4.3.3.3 Method 3

The third proposed method for creating the judgment list initiates from the pooling idea. Based on the explanation in Section 3.3.3.3, in this method, the judgment list is extracted utilizing both language models. In other words, it will take advantage of the collective intelligence that came from the captured knowledge from both language models.

The purpose is to attribute Method 3 with the generalization derived from *"gpt2-large"* alongside the perfect similarity match inherits from "*msmarco-distilbert-dot-v5"*. The compound methods are essential to cope with the shortcomings involved in the performance of recommendation systems as a subset of information retrieval engines. The final judgment list in Method 3 will be produced by computing the union of the lists generated in Method 1 and Method 2 as a shared knowledge procedure.

According to the data reported about Method 1 and Method 2 in previous sections, it is conducted that the "*msmarco-distilbert-dot-v5"* model leads to a better judgment list in Method 1 when the parameter is set to 30. While *"gpt2-large"* can achieve a more promising list of relevant items by employing Method 2 and setting the parameter value to *max(min_num_group_items)-5*. Therefore, the knowledge gathered from both language models in Method 1 and Method 2 will be pooled in Method 3.

The common items in judgment lists, created by Method 1 using "*msmarco-distilbert-dot-v5"* and Method 2 using *"gpt2-large"*, are summarized in Table 16. It should be noted that for queries 1 and 2 there is no common item between the two selected pipelines which could mean that the employed language models did not capture any common knowledge from the Science Fiction genre.

After creating the final judgment list by computing the union, that list will be used to calculate the evaluation metric for both language models' performance in the semantic search pipeline. The empirical values of the evaluation metrics calculated using Method 3 are reported in Table 17 and the code of Methods is attached in Appendix J.

As it is expected in Method 3, recall is not fixed with value 1. The reason is that the candidates come from both language models and the naive assumption of the ideal Bi-Encoder is not valid in this Method. The same reason applies to a very low value of precision.

The higher values of recall for "*msmarco-distilbert-dot-v5"* means that a greater number of items in the judgment list created by Method 3 came from Method 1 with "*msmarco-distilbert-dot-v5"* which points that there are more common items between BERTScore and Cross-Encoder for "*msmarco-distilbert-dot-v5"* language model. The indication is that "*msmarco-distilbert-dot-v5"* is more consistent in the text similarity task.

Query	**Common item**
Artificial intelligence based action movie	[]
Science fiction movie showing the future of the world	[]
Films about time traveling	['Voyage']
A movie about romance and the pain of separation	['Ninaithu Ninaithu Parthen', 'Annum Innum Ennum', 'Manasina Maathu', 'Love Actually... Sucks!', 'Faces', 'Kalgejje']
An action movie about revenge against family murder	['Sri Naga Shakthi', 'This Rebel Breed', 'Manasina Maathu', "Emily Brontë's Wuthering Heights"]
Comedy movie that contains time travel fantasy	['The Kentucky Fried Movie', 'Return of Mr. Superman']

Table 16. common items in judgment lists created by Method 1 and Method2 using "msmarco-distilbert-dot-v5" and "gpt2-large" respectively.

The reported data in Table 17 demonstrates that *"msmarco-distilbert-dot-v5"* gained the greatest value of MRR which means the best performance. This is confirmed in Method 1 and Method 2 as well. Obviously, recall and precision values for *"msmarco-distilbert-dot-v5"* surpass also. Although it should be considered that in this case, BERTScore is the best re-ranker which is different from the two other methods.

It could be concluded that since in Method 3 more generalized items coming from *"gpt2-large"* are included in the candidates' list, the BERTScore performed better in re-ranking them as it can capture more generalized similarities using "microsoft/deberta-xlarge-mnli" language model.

It would be said that the large and general-purpose language model like *"gpt2-large"* and *"microsoft/deberta-xlarge-mnli"*, which is used in BERTScore and is cited as outperforming BERT in Natural Language Understanding tasks, capture similar features. In other words, the features that are captured by *"gpt2-large"* would be considered also by the "microsoft/deberta-xlarge-mnli" model in the re-ranking process.

Language Model	Re-ranking method	MRR@6	Precision	Recall
“msmarco-distilbert-dot-v5”	Bi-Encoder	0.42	[0.04 0.12 0.42 0.06 0.04 1.00]	[0.04 0.11 0.38 0.05 0.04 0.91]
“msmarco-distilbert-dot-v5”	Cross-Encoder	0.35	[0.04 0.12 0.42 0.06 0.04 1.00]	[0.04 0.11 0.38 0.05 0.04 0.91]
“msmarco-distilbert-dot-v5”	BERTScore	**0.71**	[0.04 0.12 0.42 0.06 0.04 1.00]	[0.04 0.11 0.38 0.05 0.04 0.91]
“gpt2-large”	Bi-Encoder	0.02	[0.00 0.00 0.04 0.00 0.00 0.10]	[0.00 0.00 0.04 0.00 0.00 0.09]
“gpt2-large”	Cross-Encoder	0.33	[0.00 0.00 0.04 0.00 0.00 0.10]	[0.00 0.00 0.04 0.00 0.00 0.09]
“gpt2-large”	Bi-Encoder	0.07	[0.00 0.00 0.04 0.00 0.00 0.10]	[0.00 0.00 0.04 0.00 0.00 0.09]

Table 17. Models’ evaluation employing the judgment list created by Method 3

4.3.4 Natural Language Understanding

One of challenges in NLU is the ambiguity (polysemy). If a word like ‘driver’ is viewed, it can bring up many potential meanings including vehicle driver, software that enables the hardware to work, screwdriver, or the impetus for pushing something forward. So, the context can help to get the unique meaning that would be captured in the form of contextual embedding by using the language models.

Understanding the context in natural language understanding can be divided into two categories, understanding the user as well as understanding the domain. To understand the domain fine-tuning the model based on the domain specific data would be helpful but understanding the user’s intent depends on the ability of the employed language model on NLU tasks and boosting the system with user interactions and feedback which is beyond the scope of this research.

According to Section 3.3.4 discussion, some of the test queries are designed to put identical meanings and intent in different words in order to provide background for the Natural Language

Understanding capacity of the applied language models. For instance, queries 1 and 2 are attributed to this feature. The performance of the "*msmarco-distilbert-dot-v5*" followed by the Cross-Encoder re-ranker for queries 1 and 2 are reported in Tables 18 and 19. The "*msmarco-distilbert-dot-v5*" followed by the Cross-Encoder as the re-ranker is selected because based on information provided in the previous section it functions appropriately.

Query 1: Artificial intelligence based action movie
Query 2: Science fiction movie showing the future of the world

Cross-encoder Re-rank	Genre	Title
1	ScienceFiction	D.A.R.Y.L
6	Adventure	The Legend of Wisley
1	ScienceFiction	Ra.One
13	Action	Universal Soldier: The Return
37	ScienceFiction	Evolver
14	ScienceFiction	Transcendence

Table 18. Results for query 1

Cross-encoder Re-rank	Genre	Title
14	ScienceFiction	Time chasers
47	ScienceFiction	Transcendence
10	ScienceFiction	Lost In Space
19	ScienceFiction	Westworld
9	Action thriller	Aa Dekhen Zara
39	ScienceFiction	Alien form L.A.

Table 19. Results for query 2

The common item is *"Transcendence"* with same selected chunk for both queries which would indicate that the common intent in both queries is captured and matched as it is expected. The provided items for each query contain relevant items and fewer irrelevant ones. In addition, the difference between the results for queries could point out that the difference in the queries' meaning is captured also. The Selected chunk from *"Transcendence"* is *"Bree threaten kill max upload virus explain power heal Evelyn physical body upload virus"*. The similarity matrix for this chunk with each query is illustrated in Figures 44 and 45. Although the matrices are not perfectly explainable, the results show that the Semantic Text Similarity (STS) task is well aligned with NLU's purpose.

It is worth to be noted that *"Transcendence"* is retrieved as the common item by both BERTScore and Cross-Encoder as re-rankers which is discussed in Section 4.3.2 but in that situation the selected chunk by each re-ranker is different.

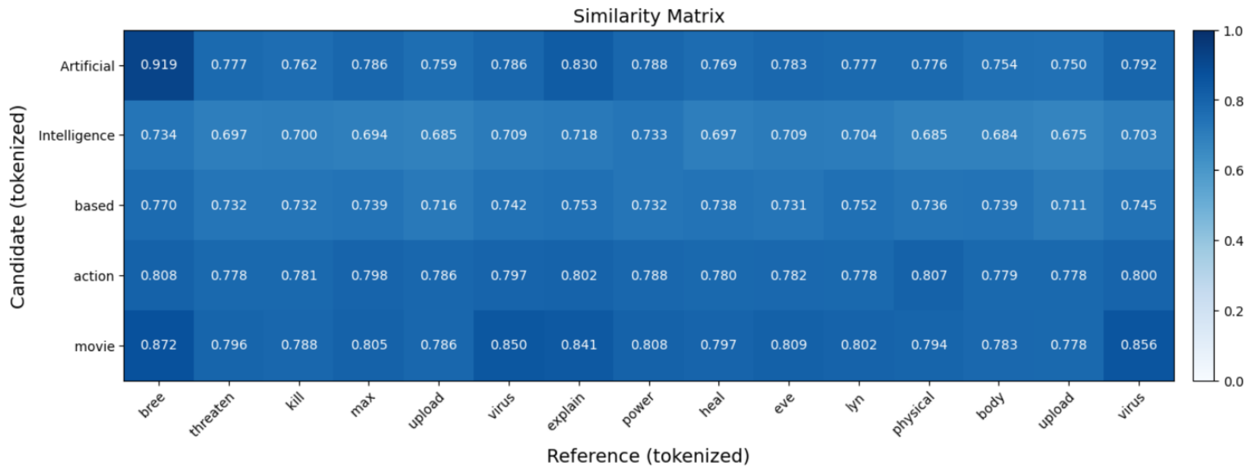


Figure 44. Similarity matrix for the selected chunk of "Transcendence" and query1

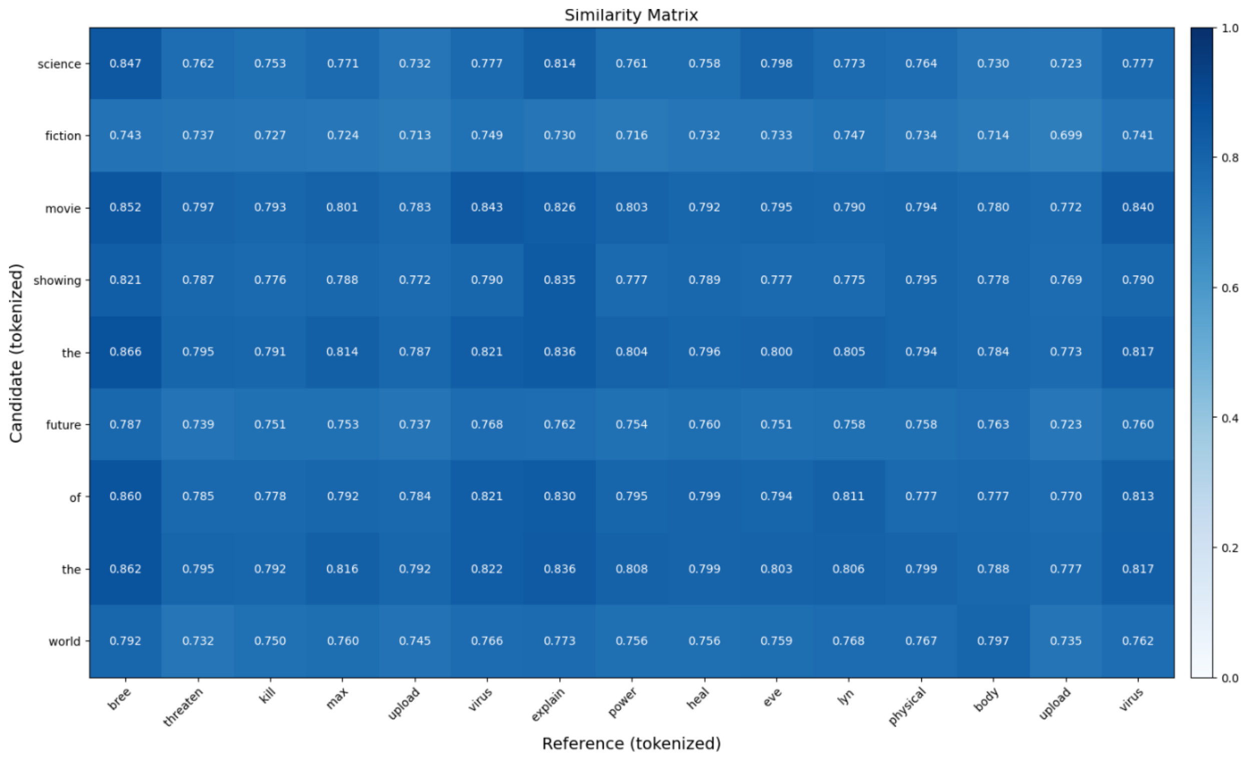


Figure 45. Similarity matrix for the selected chunk of “Transcendence” and query

Corresponding to all results documented in Section 4.3.3, all proposed methods agree on the conclusion that “*msmarco-distilbert-dot-v5”* happens to perform better than the “gpt2-large” in the semantic text similarity tasks like semantic search and recommendation systems as two types of information retrieval systems.

Methods 1 and 2 demonstrate that Cross-Encoder achieves a better re-ranking result than BERTScore However, Method 3 results show that BERTScore is a better choice with “*msmarco-distilbert-dot-v5”,* and Cross-Encoder surpasses greatly the BERTScore when it follows the “gpt2-large” embeddings. To clear the vision of the results it should be considered that *“gpt2-large”* is a large general-purpose language model with 774M parameters while “*msmarco-distilbert-dot-v5”* is a smaller language model with only 66M parameters fine-tuned for semantic text similarity. In addition, in the embedding generation phase, the input text is truncated to the maximum number of tokens that the corresponding language model can handle. Since *“gpt2-large”* can handle a longer sequence of tokens rather than “*msmarco-distilbert-dot-v5”*, the contextual information extracted and stored in embedding utilizing this model would be distinct.

According to the results documented in Appendices L and K, it seems that "*msmarco-distilbert-dot-v5*" which is well-suited for semantic textual similarity tasks is more consistent in NLU regarding user intent capturing in counterpart queries.

4.3.5 The results of a fined-tuned model on the utilized dataset

In the context of semantic search, fine-tuning is essential to accomplish tasks related to discovering text similarity (Grainger et al., 2021) which is verified by the reported data conducted in this research by applying two different language models, one is a small distilled version of the BERT model fine-tuned for semantic text similarity task and the other one a large general-purpose language model.

The major obstacle in many real-world data-driven solutions is the absence of annotated data or the high cost of providing labels. As a result, fine-tuning would not be possible. Regarding this fact creating a judgment list as a list of relevant data, where being relevant or irrelevant is the label, is sought as one of the objectives of the current research.

The labeling methods that are introduced in Section 3.3.3 are mainly proposed in this research to provide the ground truth to calculate the model performance while they can be utilized to fine-tune the pre-trained models on this dataset as well. The idea of utilizing a Cross-Encoder, BERTScore, or a combination of both to produce annotated data is similar to the innovation proposed by Chiu and Shizato (2022) to refine the human-annotated training data for Bi-Encoder models using a Cross-Encoder model. The idea should be explored in separate research in the future.

Thakur et.al. (2021) proposed a solution to resolve the absence of annotated data in the evaluation of Information Retrieval Models. They introduced ***GenQ*** which is an unsupervised domain-adaption approach for dense retrieval models using synthetic queries. Inspired by this work, Verma (2021) applied the idea of generating synthetic queries to fine-tune the SBERT for the same dataset that is used in this research. In this approach, for each chunk of the movie plots, 5 synthetic queries will be generated to represent the information as questions. Then, this extracted knowledge will be used to fine-tune the model. The *MultipleNegativesRankingLoss* is used for fine-tuning since it is a great loss function if you only have positive pairs, for example, only pairs of similar texts like pairs of paraphrases, pairs of duplicate questions, pairs of (query, response), or pairs of (source_language, target_language).

The reported results of the "*msmarco-distilbert-base-dot-prod-v3*" model after fine-tuning are summarized in Table 20.

Comparing Table 20 with Table 6 shows that fine-tune made more satisfying results.

The only common item in Table 20 with reported results in this research is the *"Crosstalk"* and *"Remote Control"*. *"Crosstalk"* is retrieved by *"gpt2-large"* for query 1 but *"Remote Control"* is matched with query2 with both employed models in this research.

Query	**Model**	**Positional rank**	**Title of the movie**
Query 1: "Artificial intelligence based action movie"	***Fine-tune*** *msmarco-distilbert-base-dot-prod-v3*	1	Remote Control
		2	Civic Duty
		3	Computer Chess
		4	Armed Response
		5	Crosstalk
Query 4: "A movie about romance and the pain of separation "	***Fine-tune model***	1	Hridayer Shabdo
		2	Manasina Maathu
		3	Murali Meets Meera
		4	Anumati
		5	Idhayam

Table 20. Bi-Encoder results using "msmarco-distilbert-base-dot-prod-v3" after fine-tuning(Verma, 2021)

Among the results of the fine-tuned model in Table 20 for query 4, *"Hridayer Shabdo"*, *"Manasina Maathu"*, and *"Murali Meets Meera"* are appeared in the reported results in this research as well (the results of this research for all queries are documented in Appendices K and L). *"Hridayer Shabdo"* is retrieved as a matched item to query 4 by *"gpt2-large"* followed by

Cross-Encoder. *"Manasina Maathu"* is suggested as a match to queries 5 and 6 by *"gpt2-large"* followed by BERTScore re-ranker and is mentioned as a matched item with queries 2, 3, 4, 5, and 6 by "*msmarco-distilbert-dot-v5"* followed by BERTScore. *"Murali Meets Meera"* is retrieved by *"gpt2-large"* followed by BERTScore in response to queries 4 and 6.

These comparisons illustrate that *"gpt2-large"* is more compatible with the fine-tuned version of the "*msmarco-distilbert-base-dot-prod-v3*" model, especially for query 4.

In general, the language models belonging to the *GPT* family are trained to predict the next token in a sequence based on the preceding tokens and they are unidirectional which means they can only consider the tokens to the left of the current token to capture the context. Consequently, It seems that the *"gpt2-large"* cannot surpass the DistilBERT model in tasks that require a deeper understanding of context and semantics even though it has more capacity regarding huger number of parameters.

On the other hand, it should be considered that the *BERT* model in general is a bidirectional model which means it can take both the preceding and succeeding tokens in capturing the context. It may be concluded that this characteristic besides the "masked language model" objective followed in the training process makes it more powerful in contextual understanding and semantic similarities. The *BERT* employs only the encoder part of the *Transformers* architecture while the *GPT* model utilizes just the decoder. Consequently, they would have different capacities and potentials in various task categories that should be involved in making decisions on the proper model for the downstream task in addition to the quantified attributes like the number of parameters and input tokens.

The opportunity of taking advantage of the characteristics of both types of models like the ability of text generation of *GPT* and the power of *BERT* in semantic understanding at the same time should be regarded in special situations likewise Method 3 in providing annotated data.

5. Conclusion and Discussion

Search engines and recommendation systems are various types of information retrieval (IR) that can be enhanced with a profound understanding of the specific domain. These systems aim to learn extensively from the domain by leveraging Language Models (LMs) as a valuable tool to capture contextual relationships and conceptually align the query intent with the ingested content.

Model selection plays a crucial role in proposing proper solutions to real-world problems regarding user satisfaction. This satisfaction would be committed considering several perspectives including desired results, low latency, and sustainability which points consuming resources as less as possible.

In this research some of prominent constraints in model selection is studied regarding the objectives aligned with user satisfaction. These constraints form the comparison baseline including applying a task specific fine-tuned model or a general model, the number of parameters which determines the learning capacity of the model, and the amount of time it takes to capture the information as well as serving the response. As fine-tuning a large language model would be too expensive regarding time and resource consumption, in this research a small, fine-tuned model, "*msmarco-distilbert-dot-v5*", is compared to a large general-purpose model, "*gpt2-large*".

The overall performance could be improved by ranking and furthermore re-ranking the outcomes of the similarity search. In this scenario, Cross-Encoder and BERTScore proved to be proficient at identifying the texts with highest semantic similarity, surpassing the Bi-Encoder algorithm. However, these methods are more computationally demanding compared to the Bi-Encoder. To make the most of their benefits, they are usually used as re-rankers on a smaller set of candidates. In this research, a comparison between both models is conducted, and in many cases, the Cross-encoder demonstrated superior performance over the BERTScore. Nonetheless, further investigation could focus on optimizing the parameter settings for the size of input chunks to achieve even more accurate outcomes. This is a further exploration in improving the accuracy of retrieved items in creating semantic search or in general retrieval system using LLMs' capabilities not only in retrieval phase but also in ranking operation. The objective of the first research question is fulfilled in this combinational utilization of LLMs.

In current research, language models are also used in the role of decision maker assistant as they can determine how similar two sentences are. Therefore, they are capable to assist in creating annotation which is explored in second research question. The Cross-Encoder and BERTScore, employed as similarity scoring methods, can be incorporated into an automatic evaluation pipeline for assessing the performance of the information retrieval system. When combined, they offer a broader perspective on approximating human judgment, serving as a viable alternative for labels in calculating evaluation metrics like Mean Reciprocal Rank (MRR) in scenarios where annotated data is unavailable. Moreover, they can be employed to supply annotated data for fine-tuning the model using domain-specific data. This is especially beneficial when the fine-tuned model significantly improves performance, compared to the resource-intensive process of providing annotations and re-training the model.

As discussed, the selection of the model is a critical aspect of ensuring the successful implementation of an information retrieval system. It serves as the core component responsible for capturing intent and context, directly impacting system performance. Additionally, it is the most computationally intensive part of the MLOps pipeline, influencing inference speed. These factors are crucial in meeting user expectations and desires. Consequently, the primary objective of this research is met as the interrelated factors involved in designing a semantic search using language models to effectively capture context and relationships with the lowest resource consumption are emphasized and analyzed.

Semantic search application can be developed and deployed as fully managed pipelines within any MLOps framework. In response to the third research question, Jina AI, among various available MLOps frameworks, is chosen as an open-source platform to develop and deploy the designed application in this research as an end-to-end solution that can be easily adapted to any other use cases by applying the same pipeline to the related dataset. It should be mentioned that the language model with the best performance and setting according to the results represented in Section 4.3 would be used in the application development.

According to the analyzed results proposed in the current experimental research, in the semantic text similarity downstream task the fine-tuned model with fewer parameters, "*msmarco-distilbert-dot-v5*", appears in better performance compared to the large general pre-trained model, "*gpt2-large*", even though it holds much higher number of parameters and learning capacity. In real-world use cases, the smaller model can be utilized on the edge, delivering satisfying performance with low latency and less resource consumption. What is aimed in the fourth research question is covered in utilizing these language models in developing the semantic search pipeline. The results indicate that the business objectives and requirements could be met less expensively by considering the effective aspect precisely and responsively. However, it is valuable to fine-tune the large model in the future as well to provide a clearer condition for comparison.

The analyzed results in comparison between similar queries articulated in different words, in response to the target of fifth research question, shows that Natural Language Understanding task in well aligned with text similarity and the explored language model abilities and affecting assumptions in the current research are applicable for both tasks.

In the current research, the input provided to the language model for generating embeddings is limited to the maximum number of tokens that can be processed by the utilized model. However, a potential area for future investigation is to explore the impact of splitting the input into smaller chunks with sizes that align with the model's token capacity, in order to assess its effect on the performance of the models. In the experimental setup applied for both re-rankers, the number of tokens used to split the input text into smaller chunks is one of the parameters. This number is selected through empirical testing of larger and smaller values. However, it is essential to note that this chosen number is significantly smaller than the number of tokens that the employed language models can handle. This specifies that the contextual capacity of the models might be compromised, which could have a considerable impact on their performance. Thus, conducting further investigation to verify this observation would be valuable.

The importance of some columns in dataset is revealed within data exploratory analysis. For example, the field “Genre” in dataset determines the distribution of the Plots and results. In addition, it contains some hidden information regarding each movie by its nature. Therefore, it would be beneficial to unleash this power to enhance the performance of the search by considering it in the pipeline and explore how it will influence in enhancing search performance. It can be used for narrowing down the results as well as data augmentation, where Genre prediction could be employed to fill in missing values within the dataset.

There are more aspects that can be explored further in conjunction with the proposed pipeline in this research. For instance, considering an augmented design including personalization through users’ interactions, utilizing knowledge extraction techniques like Named Entity Recognition (NER), Text Classification, and so on could be beneficial to deliver a more robust search experience.

Challenges like aligning the users’ intent where the same intent is conveyed in different words, low precision and high recall, and lack of experimental tests has been involved in search application development for a long time. Some of these challenges are explored in more detail during the current research and some less. There are various reasons that prevents more profound exploration including time limitation, requiring more expertise and knowledge to discuss and analyze more deeply than what is done during this research, and limited free processing units and memory available for big data analysis. So, the open questions relating to discussed aspects could be covered in future research.

Additionally, the idea of utilizing a Cross-Encoder, BERTScore, or a combination of them to generate annotated data for fine-tuning pre-trained models can be further explored in future research. Consequently, the results can be compared with the ***GENQ*** technique (Thakur et al., 2021).

This search pipeline can be utilized in Retrieval Augmented Generation applications to narrow down the contents that should be fed into the generating model like ChatGPT to make it capable to answer the questions that are asked from private documents for example the handbook of a special tool or the newly released information that it is not trained on to reduce the hallucination in generation model. This use case is also a further step to be developed on top of the proposed one.

References

Briggs, J., & Carnevali, L. (n.d.). *Evaluation Measures in Information Retrieval*. Pinecone. Retrieved July 1, 2023, from https://www.pinecone.io/learn/offline-evaluation/

Bojanowski, Piotr, Grave, Edouard, Joulin, Armand, & Mikolov, Tomas. (2017, Jun 19). Enriching Word Vectors with Subword Information. arXiv. https://arxiv.org/pdf/1607.04606.pdf

Chiu, J., & Shinzato, K. (2022). Cross-Encoder Data Annotation for Bi-Encoder Based Product Matching. *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, 161–168. https://aclanthology.org/2022.emnlp-industry.16

Clough, P., & Sanderson, M. (2013). Evaluating the performance of information retrieval systems using test collections. *Information Research*, *18*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv. http://arxiv.org/abs/1810.04805

Diedrichsen, T., & Sierra, T. (2019, April 4). *Making the Case for Human Relevance Testing*. https://www.slideshare.net/o19s/haystack-2019-making-the-case-for-human-judgement-relevance-testing-tara-diedrichsen-and-tito-sierra

Foundation models. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Foundation_models&oldid=1161263920

Glavaš, DR. G. (2020, April 20). *Evaluation in Information Retrieval*. https://www.uni-mannheim.de/media/Einrichtungen/dws/Files_People/Profs/goran/10-Evaluation-FSS20.pdf

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative Adversarial Networks* (arXiv:1406.2661). arXiv. http://arxiv.org/abs/1406.2661

Grainger, T., Turnbull, D., & Irwin, M. (2021). *AI Powered Search*. MANNING. https://www.manning.com/books/ai-powered-search

guest_blog. (2020, August 31). A Simple Introduction to Sequence to Sequence Models. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/

Ham, L. (2022, August 9). Using Cross-Encoders as reranker in multistage vector search. *Using Cross-Encoders as Reranker in Multistage Vector Search*. https://weaviate.io/blog/cross-encoders-as-reranker#:~:text=Bi%2DEncoder%20models%20are%20fast,improve%20the%20vector%20search%20experience.

He, P., Liu, X., Gao, J., & Chen, W. (2021). *DeBERTa: Decoding-enhanced BERT with Disentangled Attention* (arXiv:2006.03654). arXiv. http://arxiv.org/abs/2006.03654

Khriyenko, O. (2023, spring). *Recurrent Neural Networks (RNNs)* [PDF].

Kingma, D. P., & Welling, M. (2022). *Auto-Encoding Variational Bayes* (arXiv:1312.6114). arXiv. http://arxiv.org/abs/1312.6114

Li, F.-F., Johnson, J., & Yeung, S. (2017). *Lecture 13: Generative Models*.

Liu, L., Liu, J., & Han, J. (2021). *Multi-head or Single-head? An Empirical Comparison for Transformer Training* (arXiv:2106.09650). arXiv. http://arxiv.org/abs/2106.09650

Montantes, J. (2019, June 27). *Examining the Transformer Architecture — Part 1: The OpenAI GPT 2 Controversy*. Medium. https://towardsdatascience.com/examining-the-transformer-architecture-part-1-the-openai-gpt-2-controversy-feceda4363bb

MSMARCO Models—Sentence-Transformers documentation. (n.d.). Retrieved July 24, 2023, from https://www.sbert.net/docs/pretrained-models/msmarco-v5.html

Nadeem. (2021, March 12). Encoders-Decoders, Sequence to Sequence Architecture. *Analytics Vidhya*. https://medium.com/analytics-vidhya/encoders-decoders-sequence-to-sequence-architecture-5644efbb3392

Olah, C. (2015, September 3). *Neural Networks, Types, and Functional Programming*. http://colah.github.io/posts/2015-09-NN-Types-FP/

Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing. (2018, November 2). https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html

OpenAI GPT-3: Understanding the Architecture. (2022, May 4). The AI Dream. https://www.theaidream.com/post/openai-gpt-3-understanding-the-architecture

Pérez-Agüera, J. R., Arroyo, J., Greenberg, J., Iglesias, J. P., & Fresno, V. (2010). Using BM25F for semantic search. *Proceedings of the 3rd International Semantic Search Workshop*, 1–8. https://doi.org/10.1145/1863879.1863881

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving Language*

Understanding by Generative Pre-Training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners*.

Recurrent neural network. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Recurrent_neural_network&oldid=1162017246

Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* (arXiv:1908.10084). arXiv. http://arxiv.org/abs/1908.10084

Retrieve & Re-Rank—Sentence-Transformers documentation. (n.d.). Retrieved July 12, 2023, from https://www.sbert.net/examples/applications/retrieve_rerank/README.html

Pennington, Jeffrey, Socher, Richard, & Manning, Christopher D. (2014). GloVe: Global Vectors for Word Representation.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). *DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter* (arXiv:1910.01108). arXiv. http://arxiv.org/abs/1910.01108

Sarkar, A. (2022, February 15). *All you need to know about 'Attention' and 'Transformers'—In-depth Understanding—Part 1*. Medium. https://towardsdatascience.com/all-you-need-to-know-about-attention-and-transformers-in-depth-understanding-part-1-552f0b41d021

Semantic Search—Sentence-Transformers documentation. (n.d.-a). Retrieved July 12, 2023, from https://www.sbert.net/examples/applications/semantic-search/README.html

Semantic Search—Sentence-Transformers documentation. (n.d.-b). Retrieved July 13, 2023, from https://www.sbert.net/examples/applications/semantic-search/README.html

Sennrich, R., Haddow, B., & Birch, A. (2016). *Neural Machine Translation of Rare Words with Subword Units* (arXiv:1508.07909). arXiv. http://arxiv.org/abs/1508.07909

SentenceTransformers Documentation—Sentence-Transformers documentation. (n.d.). Retrieved July 23, 2023, from https://www.sbert.net/

Shree, P. (2020, November 10). The Journey of Open AI GPT models. *Walmart Global Tech Blog*. https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2

Solanki, S. (2022, 04 01). *How to Use GloVe Word Embeddings With PyTorch Networks?* From Coderz Column: https://coderzcolumn.com/tutorials/artificial-intelligence/how-to-use-glove-embeddings-with-pytorch#1

Stanford University School of Engineering (Director). (2017, August 11). *Lecture 13 | Generative Models*. https://www.youtube.com/watch?v=5WoItGTWV54

Summary of the tokenizers. (n.d.). Retrieved July 7, 2023, from https://huggingface.co/docs/transformers/tokenizer_summary

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). *Sequence to Sequence Learning with Neural Networks* (arXiv:1409.3215). arXiv. http://arxiv.org/abs/1409.3215

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., & Gurevych, I. (2021). *BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models* (arXiv:2104.08663). arXiv. http://arxiv.org/abs/2104.08663

Tianyi. (2023). *BERTScore* [Jupyter Notebook]. https://github.com/Tiiiger/bert_score (Original work published 2019)

Title: What's GAN (generative adversarial networks), how it works? (n.d.). Retrieved July 3, 2023, from https://www.google.com/imgres?imgurl=https://www.labellerr.com/blog/content/images/2022/12/gan-banner-1.webp&tbnid=yCEDI02jejP9kM&vet=12ahUKEwiiv_Lq6_L_AhUGEBAIHRnUAmsQMygEegUIARDrAQ..i&imgrefurl=https://www.labellerr.com/blog/what-is-gan-how-does-it-work/&docid=RF3lHe2f1aCYfM&w=685&h=402&q=generative+adversarial+networks&ved=2ahUKEwiiv_Lq6_L_AhUGEBAIHRnUAmsQMygEegUIARDrAQ&sfr=vfe&source=sh/x/im/can/1

Turnbull, D. (2021, February 21). *What Is a Judgment List?* Doug Turnbull's Blog. https://softwaredoug.com/blog/2021/02/21/what-is-a-judgment-list.html

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need* (arXiv:1706.03762). arXiv. http://arxiv.org/abs/1706.03762

Verma, S. (2021, January 5). *Semantic Search with S-BERT is all you need | by Subir Verma | MLearning.ai | Medium*. https://medium.com/mlearning-ai/semantic-search-with-s-bert-is-all-you-need-951bc710e160

Wang, W., Bao, H., Huang, S., Dong, L., & Wei, F. (2021). *MiniLMv2: Multi-Head Self-Attention Relation Distillation for Compressing Pretrained Transformers* (arXiv:2012.15828).

arXiv. http://arxiv.org/abs/2012.15828

What is MLOps? (2021, December 16). Databricks. https://www.databricks.com/glossary/mlops

Yang, A. (2022, October 3). Feature-based Transfer Learning vs Fine Tuning? *Medium*. https://angelina-yang.medium.com/feature-based-transfer-learning-vs-fine-tuning-bc8fc348a33d

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2020). *BERTScore: Evaluating Text Generation with BERT* (arXiv:1904.09675). arXiv. http://arxiv.org/abs/1904.09675

Appendix

A. MSMARCO

MS MARCO (Microsoft Machine Reading Comprehension) is a large-scale dataset focused on machine reading comprehension, question answering, and passage ranking. A variant of this task will be part of TREC and AFIRM 2019.

https://github.com/microsoft/MSMARCO-Passage-Ranking/

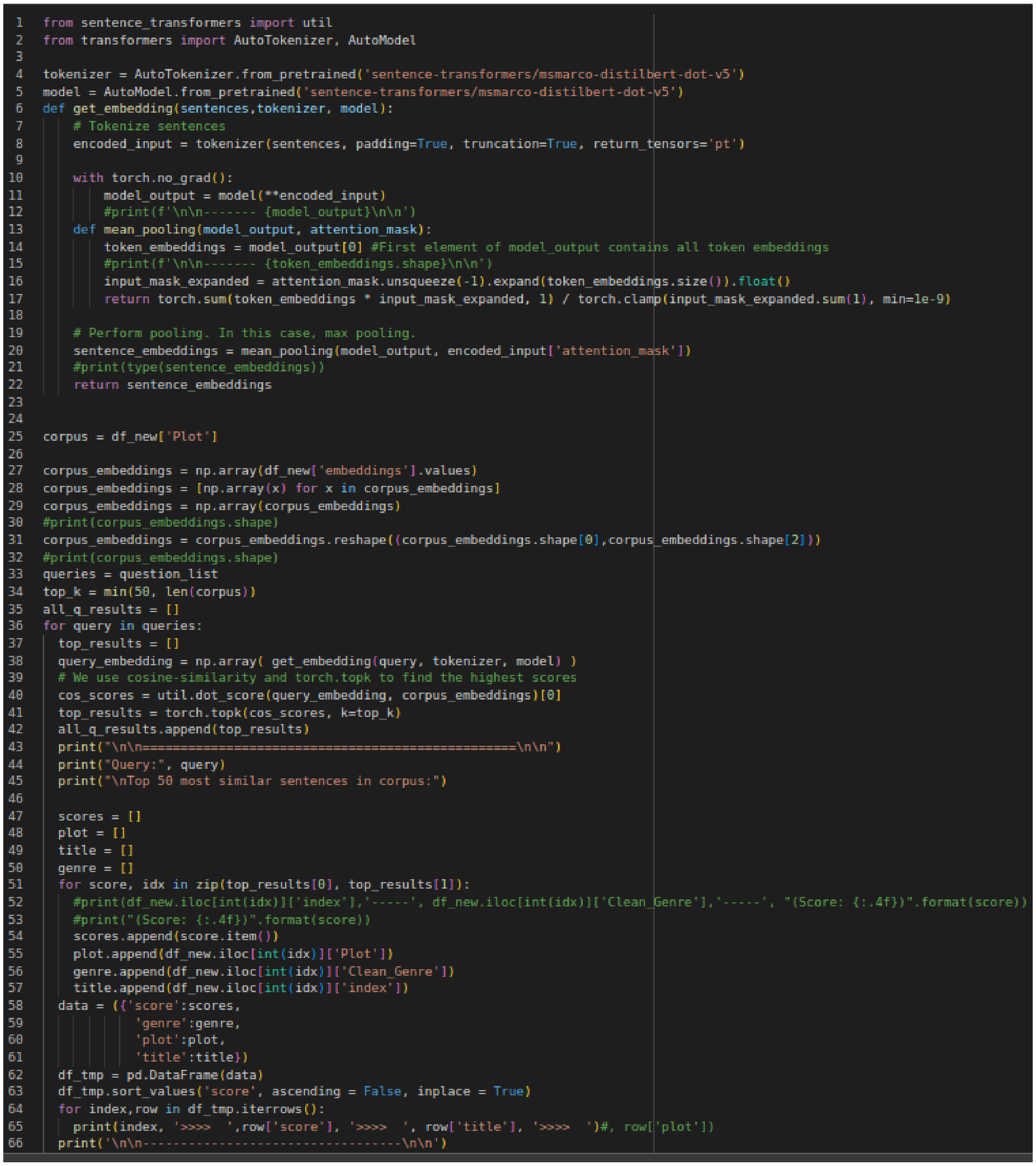
B. Movies grouped by Genre after cleaning

Genre	**Count of movies**
Drama	5643
Unknown	5584
Comedy	4191
Horror	1074
Action	1002
Thriller	897
Romance	876
Western	832
ScienceFiction	623
Comedy Drama	532
Crime	521
Adventure	472
Crime Drama	461
Romance Comedy	452
Musical	441

War	411
Film noir	326
Mystery	295
Family	201
Fantasy	193
Musical Comedy	169
Action Thriller	132
Romance Drama	130
Biography	129
Drama Romance	123
Action Comedy	114
Romantic Drama	106
Action Drama	101
Comedy Romance	101
Suspense	98
Animate	89
Family Drama	76
Romance Comedy	76
Social	76
Historical	74
Documentary	71
Crime Thriller	71
Comedy Musical	68
Drama Crime	66

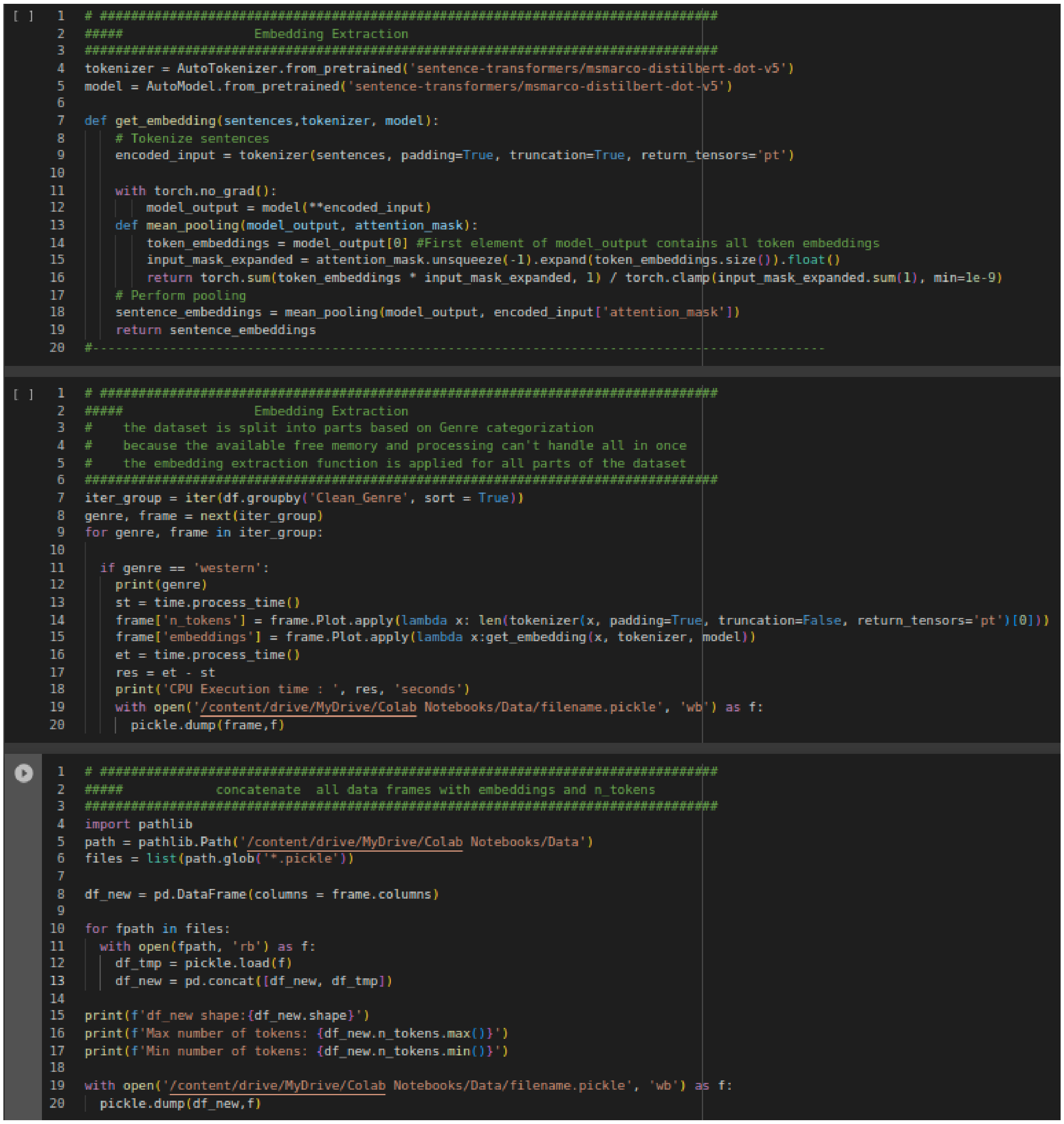
Horror Comedy	65
Action Romance	65
Historical Drama	61
Crime Comedy	56
Biopic	54
Comedy Short	52

C. Bi-Encoder



```python
from sentence_transformers import util
from transformers import AutoTokenizer, AutoModel

tokenizer = AutoTokenizer.from_pretrained('sentence-transformers/msmarco-distilbert-dot-v5')
model = AutoModel.from_pretrained('sentence-transformers/msmarco-distilbert-dot-v5')
def get_embedding(sentences,tokenizer, model):
    # Tokenize sentences
    encoded_input = tokenizer(sentences, padding=True, truncation=True, return_tensors='pt')

    with torch.no_grad():
        model_output = model(**encoded_input)
        #print(f'\n\n------- {model_output}\n\n')
    def mean_pooling(model_output, attention_mask):
        token_embeddings = model_output[0] #First element of model_output contains all token embeddings
        #print(f'\n\n------- {token_embeddings.shape}\n\n')
        input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
        return torch.sum(token_embeddings * input_mask_expanded, 1) / torch.clamp(input_mask_expanded.sum(1), min=1e-9)

    # Perform pooling. In this case, max pooling.
    sentence_embeddings = mean_pooling(model_output, encoded_input['attention_mask'])
    #print(type(sentence_embeddings))
    return sentence_embeddings


corpus = df_new['Plot']

corpus_embeddings = np.array(df_new['embeddings'].values)
corpus_embeddings = [np.array(x) for x in corpus_embeddings]
corpus_embeddings = np.array(corpus_embeddings)
#print(corpus_embeddings.shape)
corpus_embeddings = corpus_embeddings.reshape((corpus_embeddings.shape[0],corpus_embeddings.shape[2]))
#print(corpus_embeddings.shape)
queries = question_list
top_k = min(50, len(corpus))
all_q_results = []
for query in queries:
  top_results = []
  query_embedding = np.array( get_embedding(query, tokenizer, model) )
  # We use cosine-similarity and torch.topk to find the highest scores
  cos_scores = util.dot_score(query_embedding, corpus_embeddings)[0]
  top_results = torch.topk(cos_scores, k=top_k)
  all_q_results.append(top_results)
  print("\n\n================================================\n\n")
  print("Query:", query)
  print("\nTop 50 most similar sentences in corpus:")

  scores = []
  plot = []
  title = []
  genre = []
  for score, idx in zip(top_results[0], top_results[1]):
    #print(df_new.iloc[int(idx)]['index'],'-----', df_new.iloc[int(idx)]['Clean_Genre'],'-----', "(Score: {:.4f})".format(score))
    #print("(Score: {:.4f})".format(score))
    scores.append(score.item())
    plot.append(df_new.iloc[int(idx)]['Plot'])
    genre.append(df_new.iloc[int(idx)]['Clean_Genre'])
    title.append(df_new.iloc[int(idx)]['index'])
  data = ({'score':scores,
           'genre':genre,
           'plot':plot,
           'title':title})
  df_tmp = pd.DataFrame(data)
  df_tmp.sort_values('score', ascending = False, inplace = True)
  for index,row in df_tmp.iterrows():
    print(index, '>>>>  ',row['score'], '>>>>  ', row['title'], '>>>>  ')#, row['plot'])
  print('\n\n----------------------------------\n\n')
```

D. Embedding generation

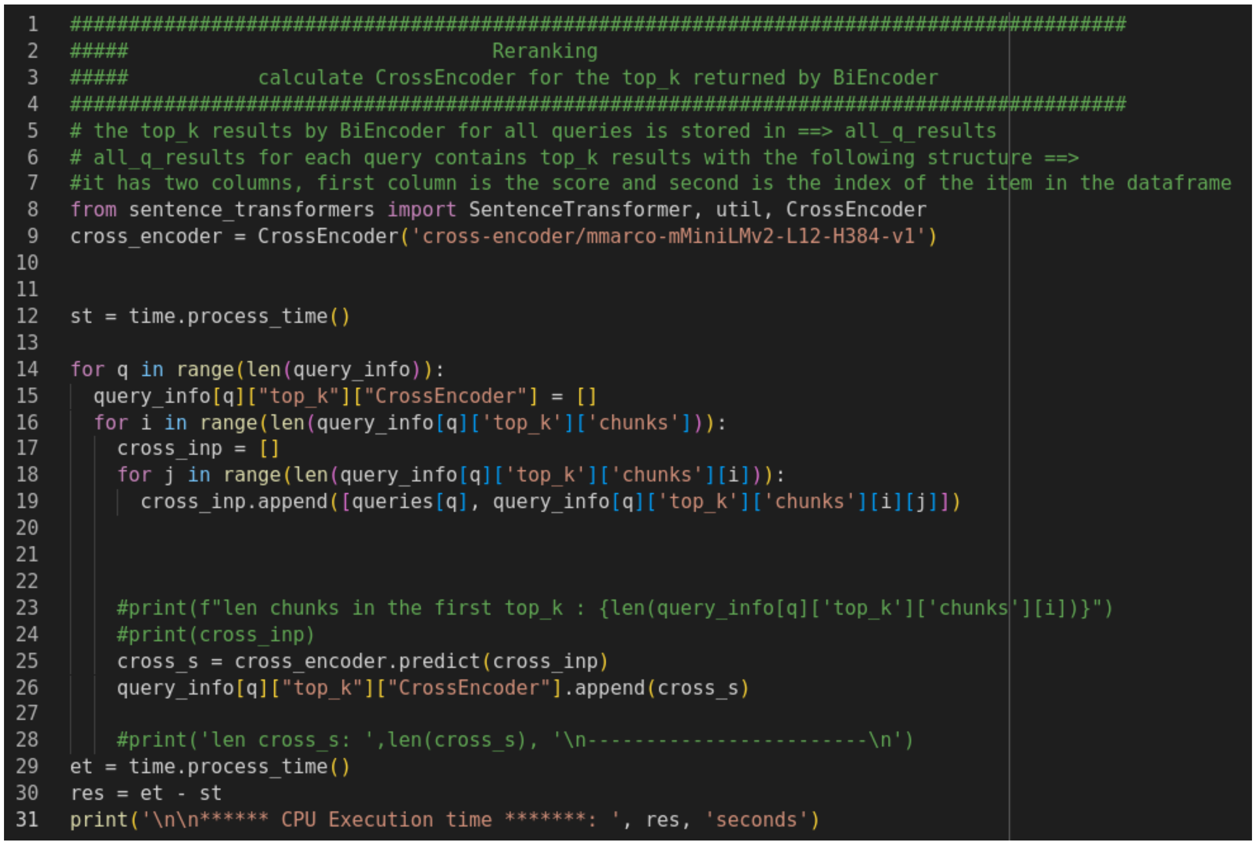


```
# ##########################################################################
#####                 Embedding Extraction
############################################################################
tokenizer = AutoTokenizer.from_pretrained('sentence-transformers/msmarco-distilbert-dot-v5')
model = AutoModel.from_pretrained('sentence-transformers/msmarco-distilbert-dot-v5')

def get_embedding(sentences,tokenizer, model):
    # Tokenize sentences
    encoded_input = tokenizer(sentences, padding=True, truncation=True, return_tensors='pt')

    with torch.no_grad():
        model_output = model(**encoded_input)
    def mean_pooling(model_output, attention_mask):
        token_embeddings = model_output[0] #First element of model_output contains all token embeddings
        input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
        return torch.sum(token_embeddings * input_mask_expanded, 1) / torch.clamp(input_mask_expanded.sum(1), min=1e-9)
    # Perform pooling
    sentence_embeddings = mean_pooling(model_output, encoded_input['attention_mask'])
    return sentence_embeddings
#-----------------------------------------------------------------------------------------------
```

```
# ##########################################################################
#####                 Embedding Extraction
#    the dataset is split into parts based on Genre categorization
#    because the available free memory and processing can't handle all in once
#    the embedding extraction function is applied for all parts of the dataset
############################################################################
iter_group = iter(df.groupby('Clean_Genre', sort = True))
genre, frame = next(iter_group)
for genre, frame in iter_group:

  if genre == 'western':
    print(genre)
    st = time.process_time()
    frame['n_tokens'] = frame.Plot.apply(lambda x: len(tokenizer(x, padding=True, truncation=False, return_tensors='pt')[0]))
    frame['embeddings'] = frame.Plot.apply(lambda x:get_embedding(x, tokenizer, model))
    et = time.process_time()
    res = et - st
    print('CPU Execution time : ', res, 'seconds')
    with open('/content/drive/MyDrive/Colab Notebooks/Data/filename.pickle', 'wb') as f:
      pickle.dump(frame,f)
```

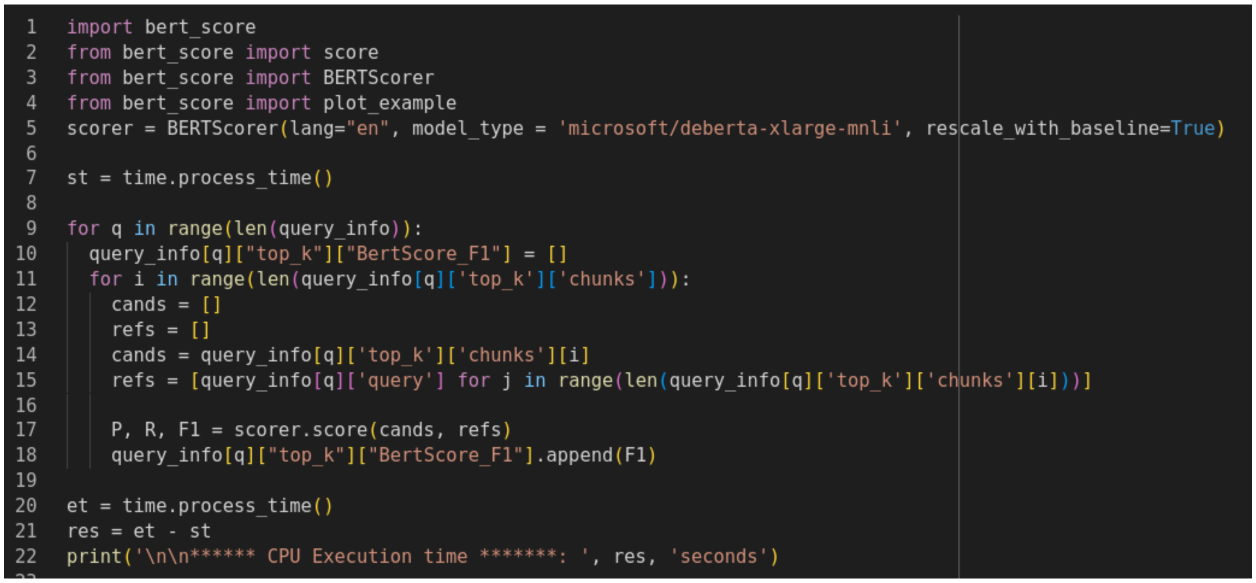
```
# ##########################################################################
#####            concatenate  all data frames with embeddings and n_tokens
############################################################################
import pathlib
path = pathlib.Path('/content/drive/MyDrive/Colab Notebooks/Data')
files = list(path.glob('*.pickle'))

df_new = pd.DataFrame(columns = frame.columns)

for fpath in files:
  with open(fpath, 'rb') as f:
    df_tmp = pickle.load(f)
    df_new = pd.concat([df_new, df_tmp])

print(f'df_new shape:{df_new.shape}')
print(f'Max number of tokens: {df_new.n_tokens.max()}')
print(f'Min number of tokens: {df_new.n_tokens.min()}')

with open('/content/drive/MyDrive/Colab Notebooks/Data/filename.pickle', 'wb') as f:
  pickle.dump(df_new,f)
```

E. Cross-Encoder as Re-ranker



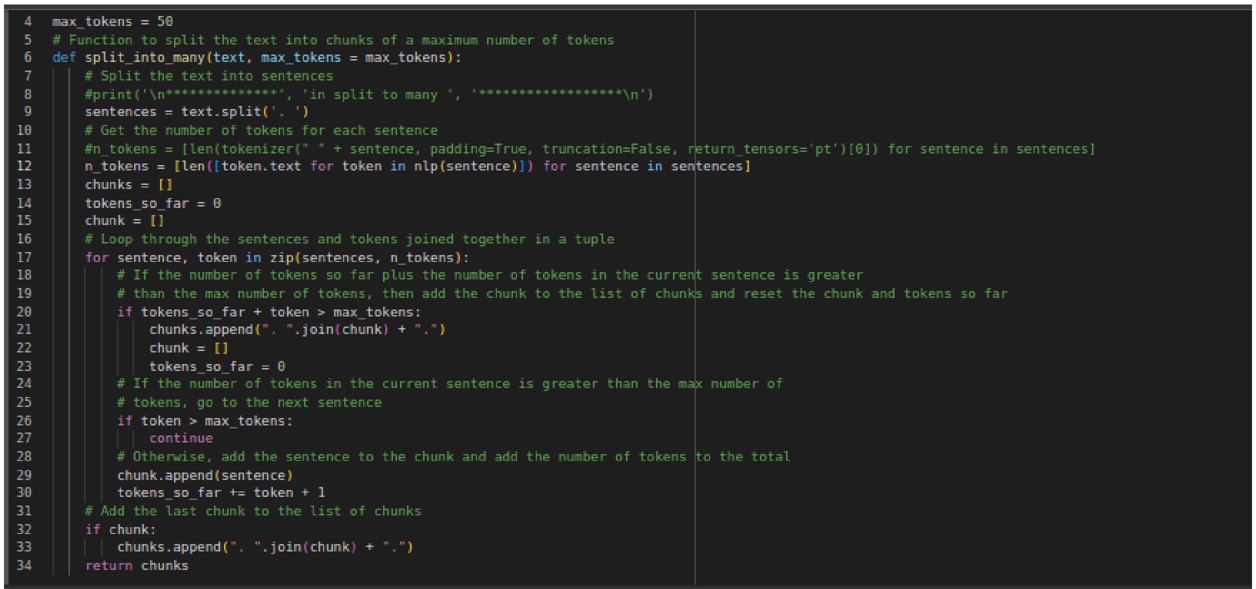
```python
##########################################################################################
#####                               Reranking
#####               calculate CrossEncoder for the top_k returned by BiEncoder
##########################################################################################
# the top_k results by BiEncoder for all queries is stored in ==> all_q_results
# all_q_results for each query contains top_k results with the following structure ==>
#it has two columns, first column is the score and second is the index of the item in the dataframe
from sentence_transformers import SentenceTransformer, util, CrossEncoder
cross_encoder = CrossEncoder('cross-encoder/mmarco-mMiniLMv2-L12-H384-v1')


st = time.process_time()

for q in range(len(query_info)):
    query_info[q]["top_k"]["CrossEncoder"] = []
    for i in range(len(query_info[q]['top_k']['chunks'])):
        cross_inp = []
        for j in range(len(query_info[q]['top_k']['chunks'][i])):
            cross_inp.append([queries[q], query_info[q]['top_k']['chunks'][i][j]])



        #print(f"len chunks in the first top_k : {len(query_info[q]['top_k']['chunks'][i])}")
        #print(cross_inp)
        cross_s = cross_encoder.predict(cross_inp)
        query_info[q]["top_k"]["CrossEncoder"].append(cross_s)

        #print('len cross_s: ',len(cross_s), '\n------------------------\n')
et = time.process_time()
res = et - st
print('\n\n****** CPU Execution time *******: ', res, 'seconds')
```

F. BERTScore as Re-ranker code

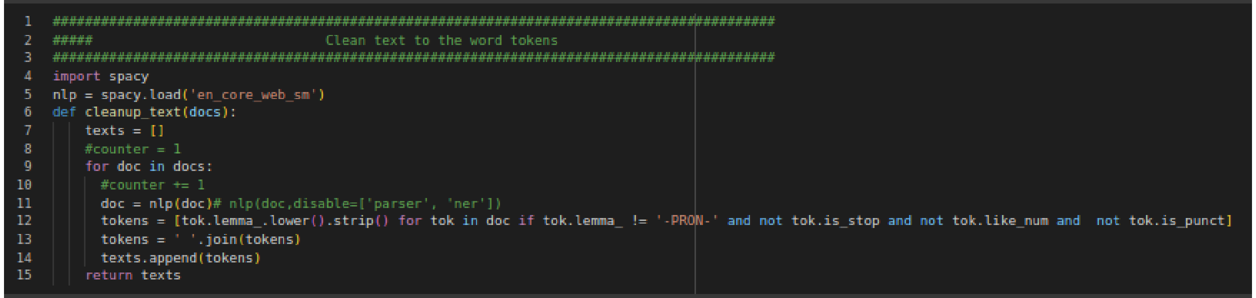


```python
import bert_score
from bert_score import score
from bert_score import BERTScorer
from bert_score import plot_example
scorer = BERTScorer(lang="en", model_type = 'microsoft/deberta-xlarge-mnli', rescale_with_baseline=True)

st = time.process_time()

for q in range(len(query_info)):
    query_info[q]["top_k"]["BertScore_F1"] = []
    for i in range(len(query_info[q]['top_k']['chunks'])):
        cands = []
        refs = []
        cands = query_info[q]['top_k']['chunks'][i]
        refs = [query_info[q]['query'] for j in range(len(query_info[q]['top_k']['chunks'][i]))]

        P, R, F1 = scorer.score(cands, refs)
        query_info[q]["top_k"]["BertScore_F1"].append(F1)

et = time.process_time()
res = et - st
print('\n\n****** CPU Execution time *******: ', res, 'seconds')
```

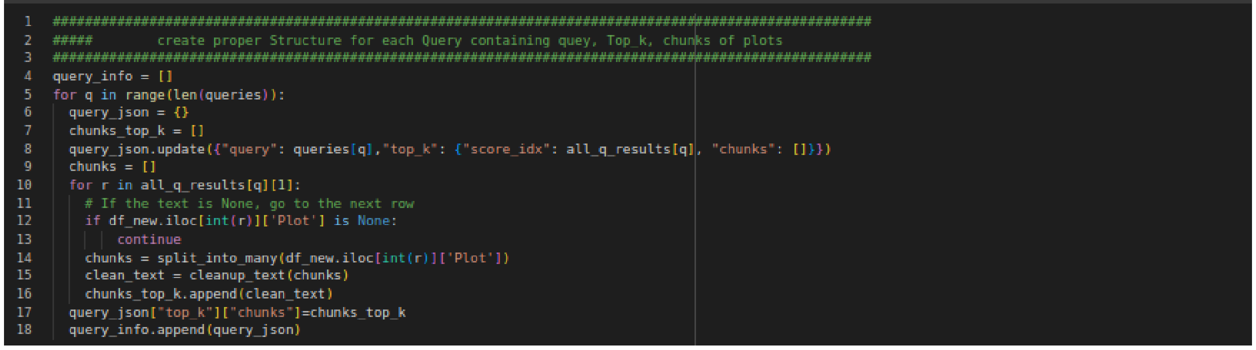
G. Splitting the Plots into chunks



```python
max_tokens = 50
# Function to split the text into chunks of a maximum number of tokens
def split_into_many(text, max_tokens = max_tokens):
    # Split the text into sentences
    #print('\n**************', 'in split to many ', '******************\n')
    sentences = text.split('. ')
    # Get the number of tokens for each sentence
    #n_tokens = [len(tokenizer(" " + sentence, padding=True, truncation=False, return_tensors='pt')[0]) for sentence in sentences]
    n_tokens = [len([token.text for token in nlp(sentence)]) for sentence in sentences]
    chunks = []
    tokens_so_far = 0
    chunk = []
    # Loop through the sentences and tokens joined together in a tuple
    for sentence, token in zip(sentences, n_tokens):
        # If the number of tokens so far plus the number of tokens in the current sentence is greater
        # than the max number of tokens, then add the chunk to the list of chunks and reset the chunk and tokens so far
        if tokens_so_far + token > max_tokens:
            chunks.append(". ".join(chunk) + ".")
            chunk = []
            tokens_so_far = 0
        # If the number of tokens in the current sentence is greater than the max number of
        # tokens, go to the next sentence
        if token > max_tokens:
            continue
        # Otherwise, add the sentence to the chunk and add the number of tokens to the total
        chunk.append(sentence)
        tokens_so_far += token + 1
    # Add the last chunk to the list of chunks
    if chunk:
        chunks.append(". ".join(chunk) + ".")
    return chunks
```

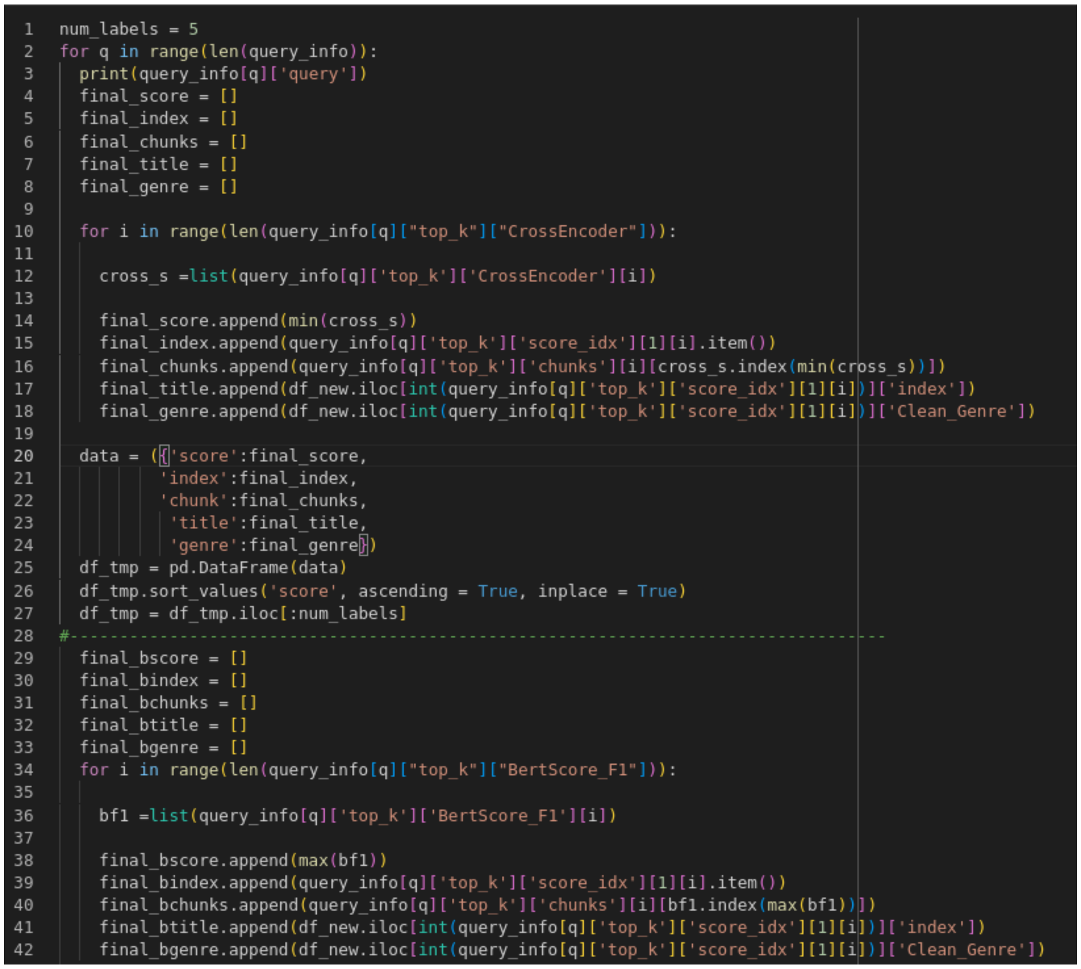


```python
##########################################################################################
#####                             Clean text to the word tokens
##########################################################################################
import spacy
nlp = spacy.load('en_core_web_sm')
def cleanup_text(docs):
    texts = []
    #counter = 1
    for doc in docs:
      #counter += 1
      doc = nlp(doc)# nlp(doc,disable=['parser', 'ner'])
      tokens = [tok.lemma_.lower().strip() for tok in doc if tok.lemma_ != '-PRON-' and not tok.is_stop and not tok.like_num and  not tok.is_punct]
      tokens = ' '.join(tokens)
      texts.append(tokens)
    return texts
```

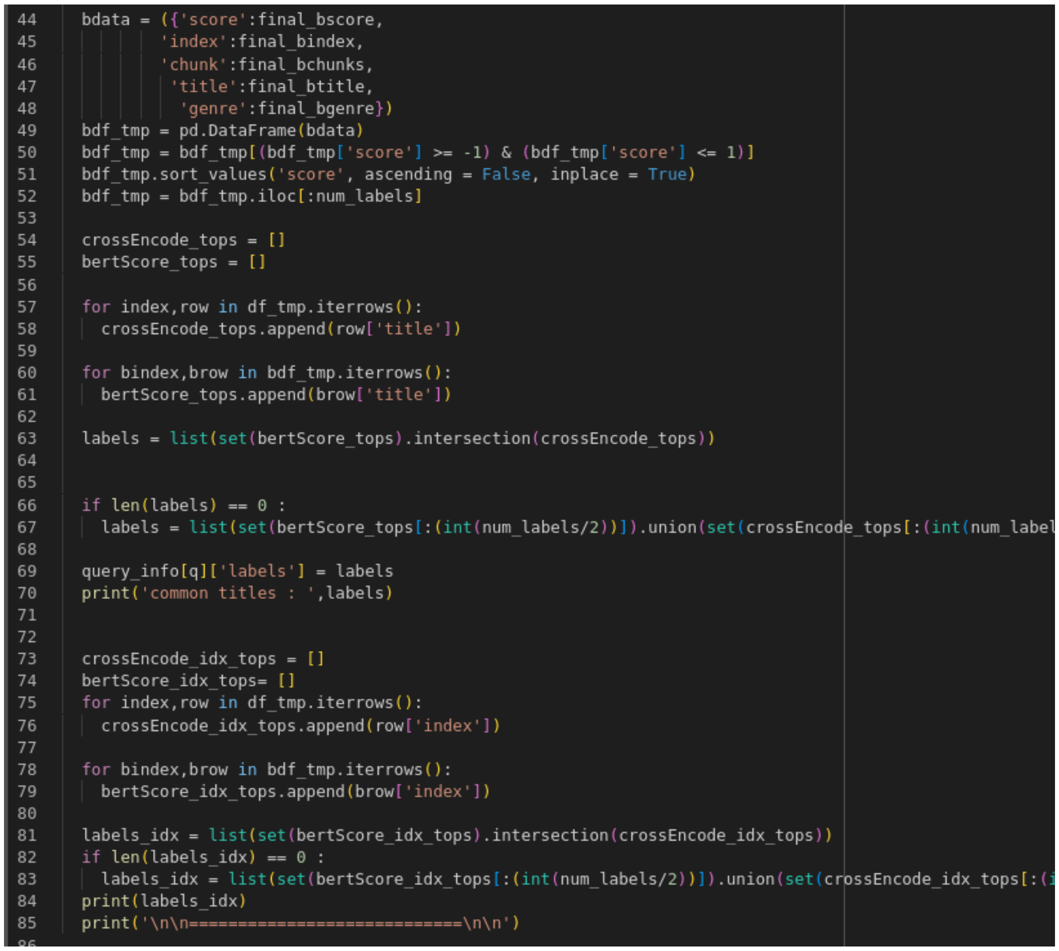


```python
######################################################################################################
#####        create proper Structure for each Query containing quey, Top_k, chunks of plots
######################################################################################################
query_info = []
for q in range(len(queries)):
  query_json = {}
  chunks_top_k = []
  query_json.update({"query": queries[q],"top_k": {"score_idx": all_q_results[q], "chunks": []}})
  chunks = []
  for r in all_q_results[q][1]:
    # If the text is None, go to the next row
    if df_new.iloc[int(r)]['Plot'] is None:
        continue
    chunks = split_into_many(df_new.iloc[int(r)]['Plot'])
    clean_text = cleanup_text(chunks)
    chunks_top_k.append(clean_text)
  query_json["top_k"]["chunks"]=chunks_top_k
  query_info.append(query_json)
```

H. Creating the judgment list with Method 1

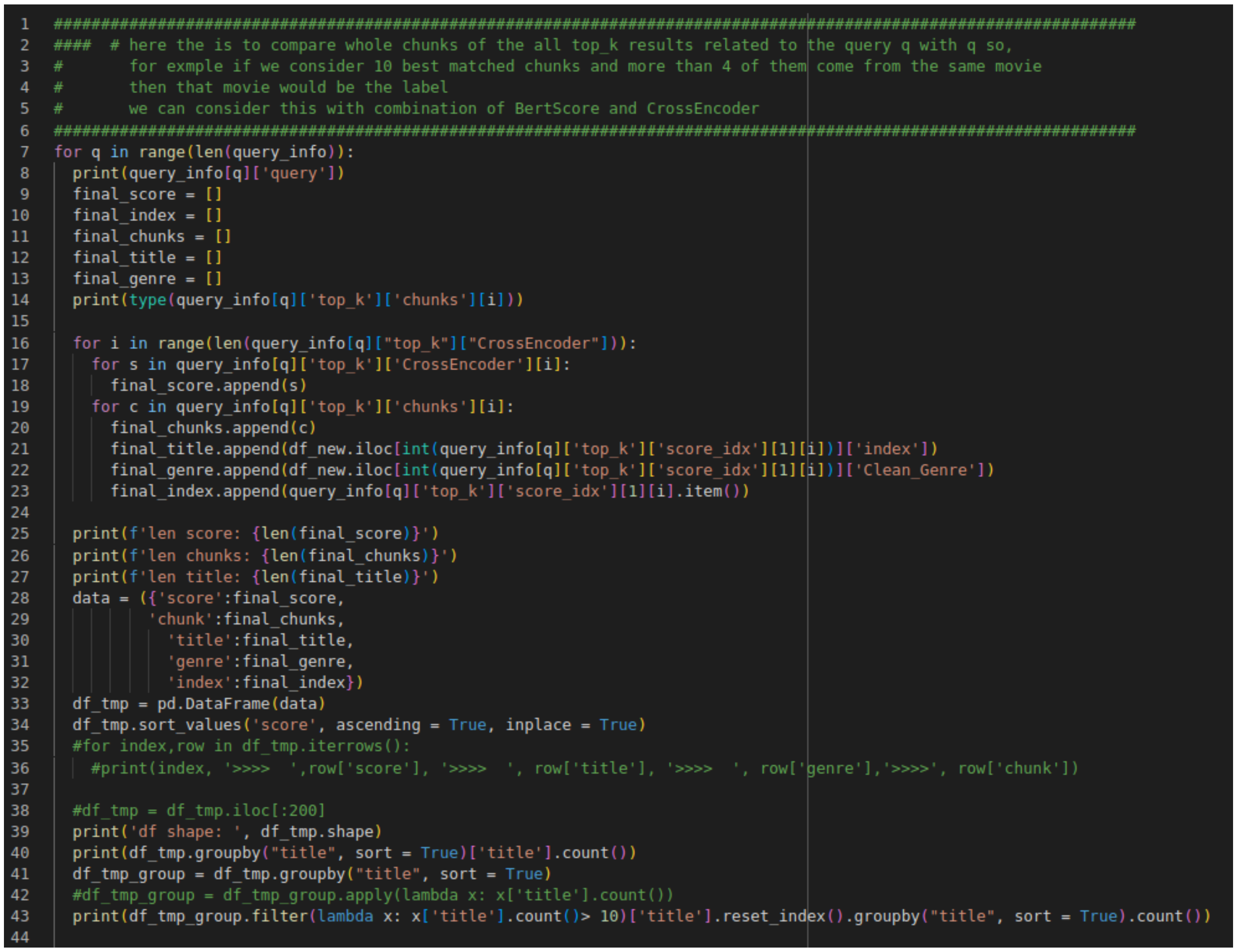


```python
num_labels = 5
for q in range(len(query_info)):
  print(query_info[q]['query'])
  final_score = []
  final_index = []
  final_chunks = []
  final_title = []
  final_genre = []

  for i in range(len(query_info[q]["top_k"]["CrossEncoder"])):

    cross_s =list(query_info[q]['top_k']['CrossEncoder'][i])

    final_score.append(min(cross_s))
    final_index.append(query_info[q]['top_k']['score_idx'][1][i].item())
    final_chunks.append(query_info[q]['top_k']['chunks'][i][cross_s.index(min(cross_s))])
    final_title.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['index'])
    final_genre.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['Clean_Genre'])

  data = ({'score':final_score,
          'index':final_index,
          'chunk':final_chunks,
           'title':final_title,
           'genre':final_genre})
  df_tmp = pd.DataFrame(data)
  df_tmp.sort_values('score', ascending = True, inplace = True)
  df_tmp = df_tmp.iloc[:num_labels]
#-------------------------------------------------------------------------------
  final_bscore = []
  final_bindex = []
  final_bchunks = []
  final_btitle = []
  final_bgenre = []
  for i in range(len(query_info[q]["top_k"]["BertScore_F1"])):

    bf1 =list(query_info[q]['top_k']['BertScore_F1'][i])

    final_bscore.append(max(bf1))
    final_bindex.append(query_info[q]['top_k']['score_idx'][1][i].item())
    final_bchunks.append(query_info[q]['top_k']['chunks'][i][bf1.index(max(bf1))])
    final_btitle.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['index'])
    final_bgenre.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['Clean_Genre'])
```

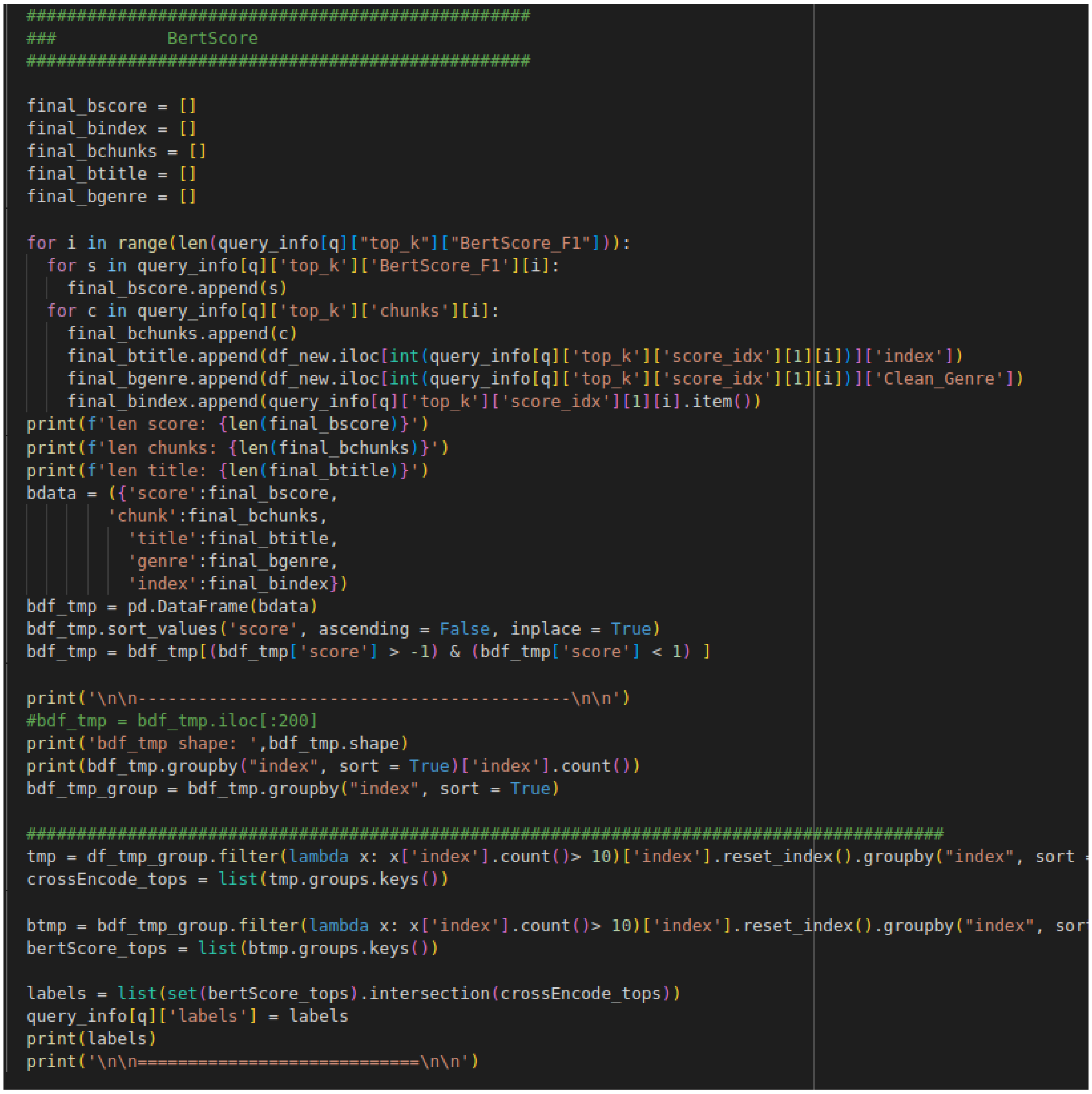


```python
bdata = ({'score':final_bscore,
        'index':final_bindex,
        'chunk':final_bchunks,
         'title':final_btitle,
          'genre':final_bgenre})
bdf_tmp = pd.DataFrame(bdata)
bdf_tmp = bdf_tmp[(bdf_tmp['score'] >= -1) & (bdf_tmp['score'] <= 1)]
bdf_tmp.sort_values('score', ascending = False, inplace = True)
bdf_tmp = bdf_tmp.iloc[:num_labels]

crossEncode_tops = []
bertScore_tops = []

for index,row in df_tmp.iterrows():
  crossEncode_tops.append(row['title'])

for bindex,brow in bdf_tmp.iterrows():
  bertScore_tops.append(brow['title'])

labels = list(set(bertScore_tops).intersection(crossEncode_tops))


if len(labels) == 0 :
  labels = list(set(bertScore_tops[:(int(num_labels/2))]).union(set(crossEncode_tops[:(int(num_label

query_info[q]['labels'] = labels
print('common titles : ',labels)


crossEncode_idx_tops = []
bertScore_idx_tops= []
for index,row in df_tmp.iterrows():
  crossEncode_idx_tops.append(row['index'])

for bindex,brow in bdf_tmp.iterrows():
  bertScore_idx_tops.append(brow['index'])

labels_idx = list(set(bertScore_idx_tops).intersection(crossEncode_idx_tops))
if len(labels_idx) == 0 :
  labels_idx = list(set(bertScore_idx_tops[:(int(num_labels/2))]).union(set(crossEncode_idx_tops[:(i
print(labels_idx)
print('\n\n============================\n\n')
```

I. create the judgment list with Method 2

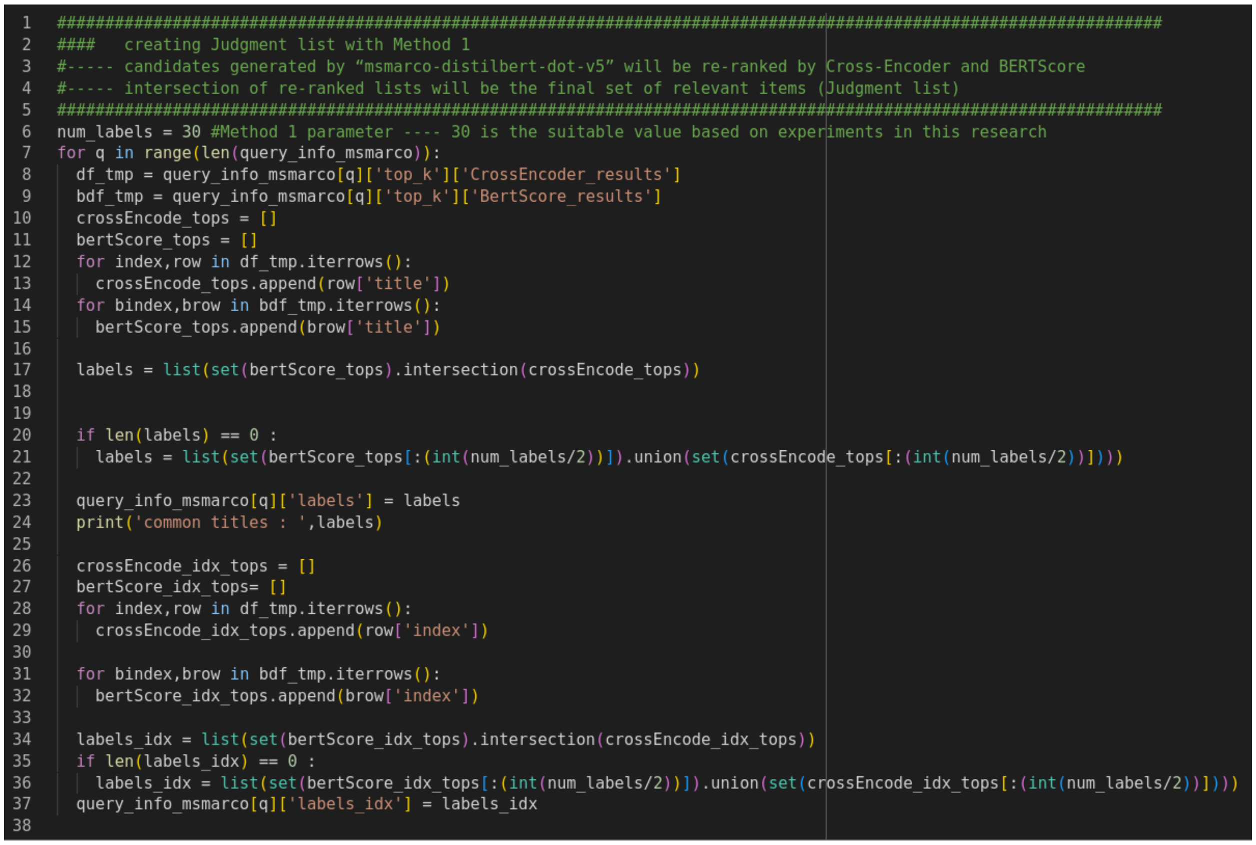


```python
##################################################################################################################
####  # here the is to compare whole chunks of the all top_k results related to the query q with q so,
#       for exmple if we consider 10 best matched chunks and more than 4 of them come from the same movie
#       then that movie would be the label
#       we can consider this with combination of BertScore and CrossEncoder
##################################################################################################################
for q in range(len(query_info)):
  print(query_info[q]['query'])
  final_score = []
  final_index = []
  final_chunks = []
  final_title = []
  final_genre = []
  print(type(query_info[q]['top_k']['chunks'][i]))

  for i in range(len(query_info[q]["top_k"]["CrossEncoder"])):
    for s in query_info[q]['top_k']['CrossEncoder'][i]:
      final_score.append(s)
    for c in query_info[q]['top_k']['chunks'][i]:
      final_chunks.append(c)
      final_title.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['index'])
      final_genre.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['Clean_Genre'])
      final_index.append(query_info[q]['top_k']['score_idx'][1][i].item())

  print(f'len score: {len(final_score)}')
  print(f'len chunks: {len(final_chunks)}')
  print(f'len title: {len(final_title)}')
  data = ({'score':final_score,
          'chunk':final_chunks,
            'title':final_title,
            'genre':final_genre,
            'index':final_index})
  df_tmp = pd.DataFrame(data)
  df_tmp.sort_values('score', ascending = True, inplace = True)
  #for index,row in df_tmp.iterrows():
    #print(index, '>>>>  ',row['score'], '>>>>  ', row['title'], '>>>>  ', row['genre'],'>>>>', row['chunk'])

  #df_tmp = df_tmp.iloc[:200]
  print('df shape: ', df_tmp.shape)
  print(df_tmp.groupby("title", sort = True)['title'].count())
  df_tmp_group = df_tmp.groupby("title", sort = True)
  #df_tmp_group = df_tmp_group.apply(lambda x: x['title'].count())
  print(df_tmp_group.filter(lambda x: x['title'].count()> 10)['title'].reset_index().groupby("title", sort = True).count())

```

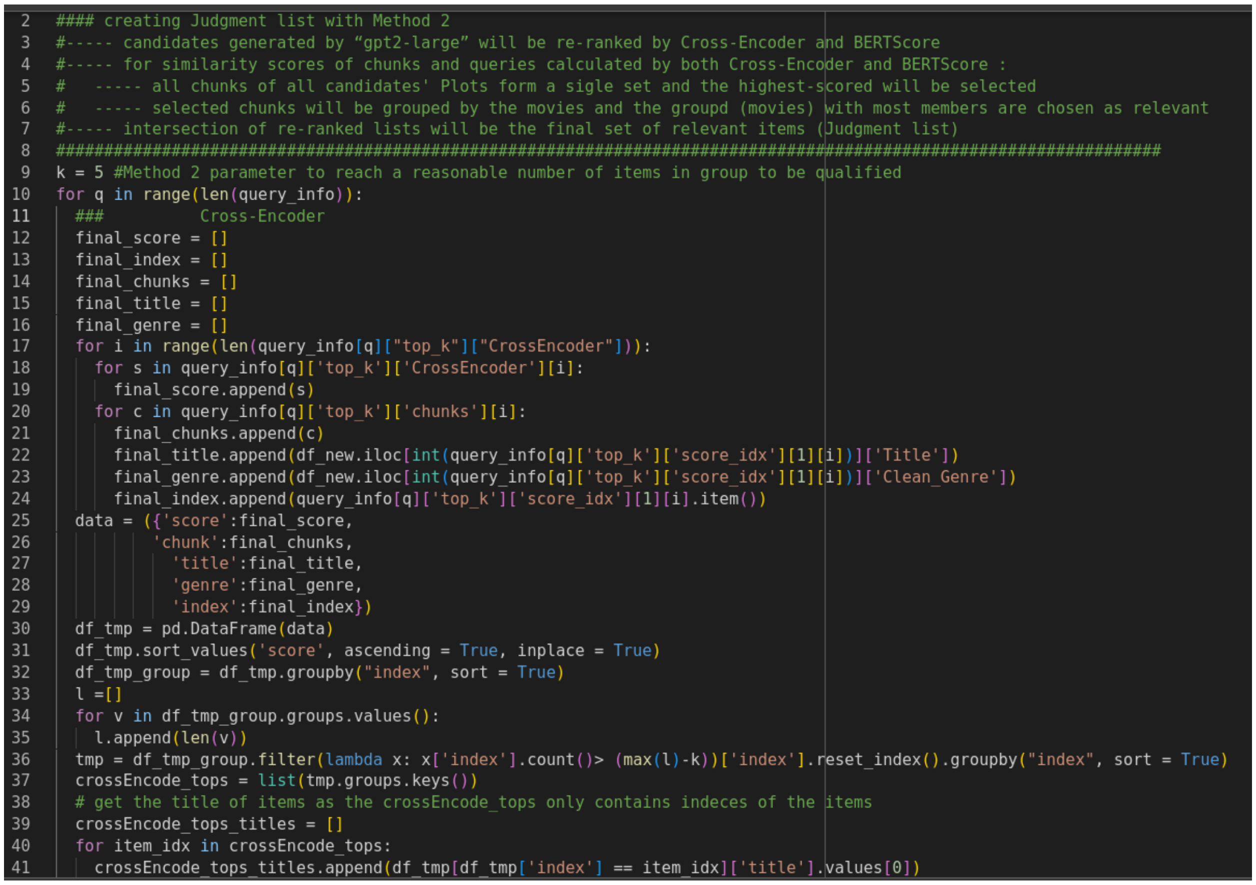


```
##################################################
###           BertScore
##################################################

final_bscore = []
final_bindex = []
final_bchunks = []
final_btitle = []
final_bgenre = []

for i in range(len(query_info[q]["top_k"]["BertScore_F1"])):
    for s in query_info[q]['top_k']['BertScore_F1'][i]:
        final_bscore.append(s)
    for c in query_info[q]['top_k']['chunks'][i]:
        final_bchunks.append(c)
        final_btitle.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['index'])
        final_bgenre.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['Clean_Genre'])
        final_bindex.append(query_info[q]['top_k']['score_idx'][1][i].item())
print(f'len score: {len(final_bscore)}')
print(f'len chunks: {len(final_bchunks)}')
print(f'len title: {len(final_btitle)}')
bdata = ({'score':final_bscore,
        'chunk':final_bchunks,
          'title':final_btitle,
          'genre':final_bgenre,
          'index':final_bindex})
bdf_tmp = pd.DataFrame(bdata)
bdf_tmp.sort_values('score', ascending = False, inplace = True)
bdf_tmp = bdf_tmp[(bdf_tmp['score'] > -1) & (bdf_tmp['score'] < 1) ]

print('\n\n-------------------------------------------\n\n')
#bdf_tmp = bdf_tmp.iloc[:200]
print('bdf_tmp shape: ',bdf_tmp.shape)
print(bdf_tmp.groupby("index", sort = True)['index'].count())
bdf_tmp_group = bdf_tmp.groupby("index", sort = True)

##############################################################################################
tmp = df_tmp_group.filter(lambda x: x['index'].count()> 10)['index'].reset_index().groupby("index", sort 
crossEncode_tops = list(tmp.groups.keys())

btmp = bdf_tmp_group.filter(lambda x: x['index'].count()> 10)['index'].reset_index().groupby("index", sor
bertScore_tops = list(btmp.groups.keys())

labels = list(set(bertScore_tops).intersection(crossEncode_tops))
query_info[q]['labels'] = labels
print(labels)
print('\n\n============================\n\n')
```

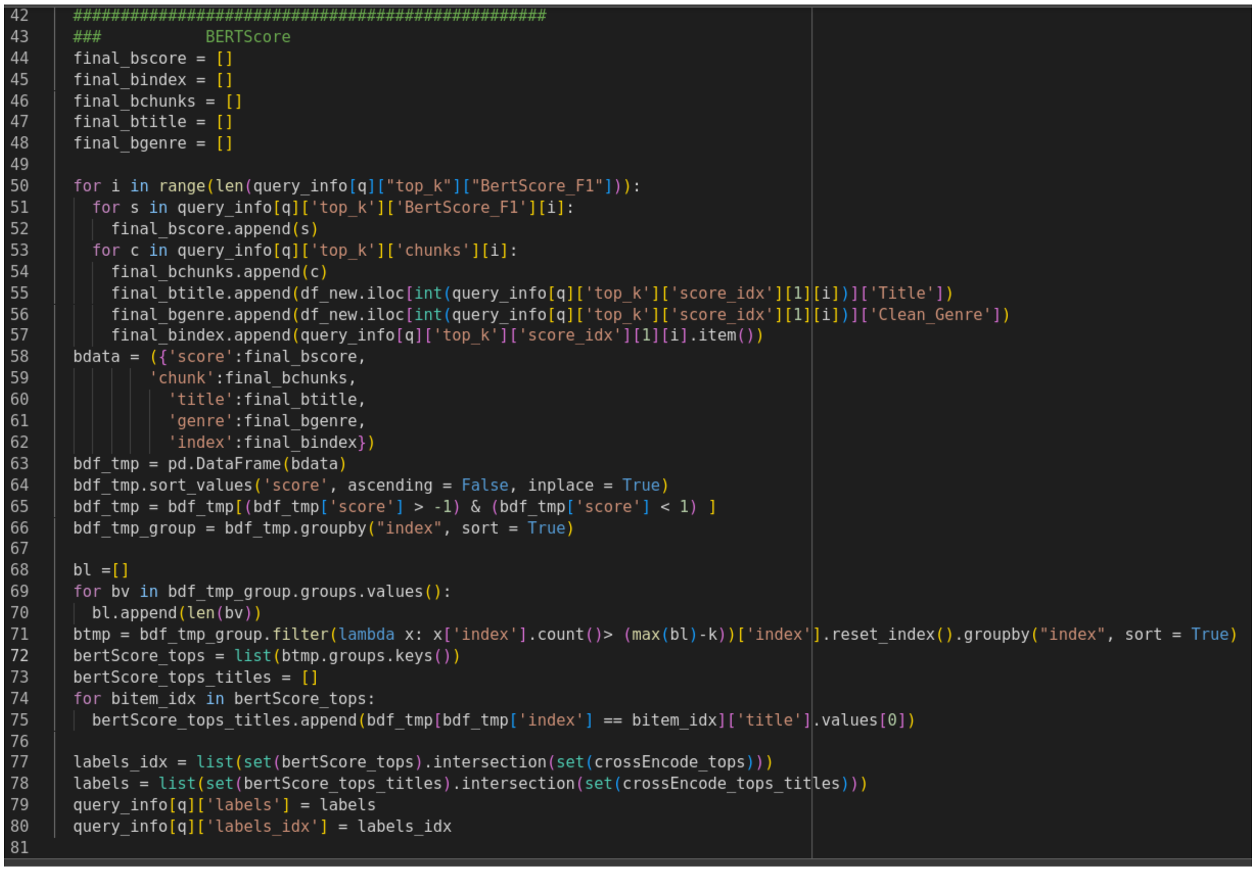
J. create the judgment list with Method 3



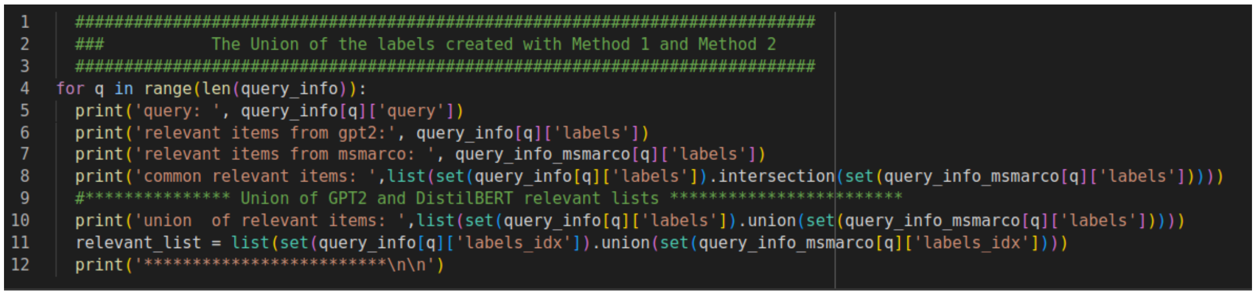
```
###################################################################################################################
####   creating Judgment list with Method 1
#----- candidates generated by “msmarco-distilbert-dot-v5” will be re-ranked by Cross-Encoder and BERTScore
#----- intersection of re-ranked lists will be the final set of relevant items (Judgment list)
###################################################################################################################
num_labels = 30 #Method 1 parameter ---- 30 is the suitable value based on experiments in this research
for q in range(len(query_info_msmarco)):
  df_tmp = query_info_msmarco[q]['top_k']['CrossEncoder_results']
  bdf_tmp = query_info_msmarco[q]['top_k']['BertScore_results']
  crossEncode_tops = []
  bertScore_tops = []
  for index,row in df_tmp.iterrows():
    crossEncode_tops.append(row['title'])
  for bindex,brow in bdf_tmp.iterrows():
    bertScore_tops.append(brow['title'])

  labels = list(set(bertScore_tops).intersection(crossEncode_tops))


  if len(labels) == 0 :
    labels = list(set(bertScore_tops[:(int(num_labels/2))]).union(set(crossEncode_tops[:(int(num_labels/2))])))

  query_info_msmarco[q]['labels'] = labels
  print('common titles : ',labels)

  crossEncode_idx_tops = []
  bertScore_idx_tops= []
  for index,row in df_tmp.iterrows():
    crossEncode_idx_tops.append(row['index'])

  for bindex,brow in bdf_tmp.iterrows():
    bertScore_idx_tops.append(brow['index'])

  labels_idx = list(set(bertScore_idx_tops).intersection(crossEncode_idx_tops))
  if len(labels_idx) == 0 :
    labels_idx = list(set(bertScore_idx_tops[:(int(num_labels/2))]).union(set(crossEncode_idx_tops[:(int(num_labels/2))])))
  query_info_msmarco[q]['labels_idx'] = labels_idx

```



```
#### creating Judgment list with Method 2
#----- candidates generated by “gpt2-large” will be re-ranked by Cross-Encoder and BERTScore
#----- for similarity scores of chunks and queries calculated by both Cross-Encoder and BERTScore :
#   ----- all chunks of all candidates' Plots form a sigle set and the highest-scored will be selected
#   ----- selected chunks will be grouped by the movies and the groupd (movies) with most members are chosen as relevant
#----- intersection of re-ranked lists will be the final set of relevant items (Judgment list)
##################################################################################################################
k = 5 #Method 2 parameter to reach a reasonable number of items in group to be qualified
for q in range(len(query_info)):
  ###          Cross-Encoder
  final_score = []
  final_index = []
  final_chunks = []
  final_title = []
  final_genre = []
  for i in range(len(query_info[q]["top_k"]["CrossEncoder"])):
    for s in query_info[q]['top_k']['CrossEncoder'][i]:
      final_score.append(s)
    for c in query_info[q]['top_k']['chunks'][i]:
      final_chunks.append(c)
      final_title.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['Title'])
      final_genre.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['Clean_Genre'])
      final_index.append(query_info[q]['top_k']['score_idx'][1][i].item())
  data = ({'score':final_score,
          'chunk':final_chunks,
            'title':final_title,
            'genre':final_genre,
            'index':final_index})
  df_tmp = pd.DataFrame(data)
  df_tmp.sort_values('score', ascending = True, inplace = True)
  df_tmp_group = df_tmp.groupby("index", sort = True)
  l =[]
  for v in df_tmp_group.groups.values():
    l.append(len(v))
  tmp = df_tmp_group.filter(lambda x: x['index'].count()> (max(l)-k))['index'].reset_index().groupby("index", sort = True)
  crossEncode_tops = list(tmp.groups.keys())
  # get the title of items as the crossEncode_tops only contains indeces of the items
  crossEncode_tops_titles = []
  for item_idx in crossEncode_tops:
    crossEncode_tops_titles.append(df_tmp[df_tmp['index'] == item_idx]['title'].values[0])
```



```python
    ##################################################
    ###           BERTScore
    final_bscore = []
    final_bindex = []
    final_bchunks = []
    final_btitle = []
    final_bgenre = []

    for i in range(len(query_info[q]["top_k"]["BertScore_F1"])):
        for s in query_info[q]['top_k']['BertScore_F1'][i]:
            final_bscore.append(s)
        for c in query_info[q]['top_k']['chunks'][i]:
            final_bchunks.append(c)
            final_btitle.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['Title'])
            final_bgenre.append(df_new.iloc[int(query_info[q]['top_k']['score_idx'][1][i])]['Clean_Genre'])
            final_bindex.append(query_info[q]['top_k']['score_idx'][1][i].item())
    bdata = ({'score':final_bscore,
            'chunk':final_bchunks,
              'title':final_btitle,
              'genre':final_bgenre,
              'index':final_bindex})
    bdf_tmp = pd.DataFrame(bdata)
    bdf_tmp.sort_values('score', ascending = False, inplace = True)
    bdf_tmp = bdf_tmp[(bdf_tmp['score'] > -1) & (bdf_tmp['score'] < 1) ]
    bdf_tmp_group = bdf_tmp.groupby("index", sort = True)

    bl =[]
    for bv in bdf_tmp_group.groups.values():
        bl.append(len(bv))
    btmp = bdf_tmp_group.filter(lambda x: x['index'].count()> (max(bl)-k))['index'].reset_index().groupby("index", sort = True)
    bertScore_tops = list(btmp.groups.keys())
    bertScore_tops_titles = []
    for bitem_idx in bertScore_tops:
        bertScore_tops_titles.append(bdf_tmp[bdf_tmp['index'] == bitem_idx]['title'].values[0])

    labels_idx = list(set(bertScore_tops).intersection(set(crossEncode_tops)))
    labels = list(set(bertScore_tops_titles).intersection(set(crossEncode_tops_titles)))
    query_info[q]['labels'] = labels
    query_info[q]['labels_idx'] = labels_idx
```



```python
  ##############################################################################
  ###           The Union of the labels created with Method 1 and Method 2
  ##############################################################################
for q in range(len(query_info)):
  print('query: ', query_info[q]['query'])
  print('relevant items from gpt2:', query_info[q]['labels'])
  print('relevant items from msmarco: ', query_info_msmarco[q]['labels'])
  print('common relevant items: ',list(set(query_info[q]['labels']).intersection(set(query_info_msmarco[q]['labels']))))
  #*************** Union of GPT2 and DistilBERT relevant lists *************************
  print('union  of relevant items: ',list(set(query_info[q]['labels']).union(set(query_info_msmarco[q]['labels']))))
  relevant_list = list(set(query_info[q]['labels_idx']).union(set(query_info_msmarco[q]['labels_idx'])))
  print('*************************\n\n')
```

K. "msmarco-distilbert-dot-v5" results for all queries re-ranked with both models

Re-ranked with Cross-Encoder

Results for Query 1

Rank	Re-rank	title	genre
0	0	D.A.R.Y.L.	ScienceFiction
6	1	The Legend of Wisley	adventure
1	2	Ra.One	ScienceFiction
13	3	Universal Soldier: The Return	action
37	4	Evolver	ScienceFiction
14	5	Transcendence	ScienceFiction
10	6	Leonard Part 6	comedy
42	7	Terminator 2: Judgment Day	ScienceFiction
5	8	Black Eagle	action
8	9	You Don't Mess with the Zohan	comedy

Results for Query 2

Rank	Re-rank	title	genre
14	0	Time Chasers	ScienceFiction
47	1	Transcendence	ScienceFiction
10	2	Lost In Space	ScienceFiction
19	3	Westworld	ScienceFiction
9	4	Aa Dekhen Zara	action thriller
39	5	Alien from L.A.	ScienceFiction
2	6	Akhil	action
23	7	The Last Starfighter	ScienceFiction
41	8	Cool World	fantasy
3	9	Remote Control	comedy

Results for Query 3:

Ran k	Re-rank	title	genre
34	0	10 MPH	documentary
23	1	Idaho Transfer	scienceFiction
5	2	Road, Movie	drama
25	3	Journey to the Center of Time	scienceFiction
11	4	Inner Sanctum	film noir
6	5	Spirit of '76	comedy
14	6	Around the Bend	drama
41	7	Esther Waters	historical drama
46	8	Private Property	crime drama
7	9	Innocents in Paris	comedy

Results for Query 4:

Rank	Re-rank	title	genre
48	0	Itlu Sravani Subramanyam	romance
4	1	Cake	drama
11	2	Ekla Akash	drama
40	3	Irreconcilable Differences	comedy
30	4	In the Cool of the Day	drama
43	5	I Shot Andy Warhol	drama
41	6	Arike (അരികെ)	romance
35	7	The Groomsmen	comedy
13	8	Mystic Pizza	comedy
21	9	Things You Can Tell Just by Looking at Her	romance

Results for Query 5:

Rank	Re-rank	title	genre
37	0	Felon	crime drama
9	1	Gunda	action comedy
43	2	The Devil's Disciple	drama
12	3	Oxygen	action
46	4	Kill Dil	romance drama
22	5	Kasoor	thriller
21	6	Andhrawala	action
40	7	10 to Midnight	action
19	8	The Horseman	thriller
38	9	Renigunta	action

Results for Query 6:

Rank	Re-rank	title	genre
20	0	3G	thriller
31	1	Return of Mr. Superman	action
28	2	Interkosmos	drama
5	3	The Legend of Wisley	adventure
40	4	Ego	comedy
39	5	Innocents in Paris	comedy
47	6	Mission to Moscow	war
8	7	Hands Up!	comedy
42	8	The Waiting Room	drama
44	9	Johnny Doesn't Live Here Anymore	romance

Re-ranked with BERTScore

Results for Query 1:

Rank	Re-rank	title	genre
14	0	Transcendence	scienceFiction
24	1	Nemesis	scienceFiction
26	2	Kantri	action
0	3	D.A.R.Y.L.	scienceFiction
4	4	Chappie	scienceFiction
37	5	Evolver	scienceFiction
32	6	Terminator Salvation	scienceFiction
48	7	Rangamati	romance
40	8	Katha	comedy drama
31	9	A Gentleman	action romance

Results for Query 2:

Rank	Re-rank	title	genre
7	0	Dr. Plonk	comedy
31	1	Impostor	scienceFiction
2	2	Akhil	action
47	3	Transcendence	scienceFiction
36	4	Unknown World	scienceFiction
24	5	Inner Sanctum	film noir
22	6	Manasina Maathu	romance
32	7	It Conquered the World	scienceFiction
41	8	Cool World	fantasy
42	9	Monster A Go-Go	scienceFiction

Results for Query 3:

Rank	Re-rank	title	genre
7	0	Dr. Plonk	comedy
31	1	Impostor	scienceFiction
2	2	Akhil	action
47	3	Transcendence	scienceFiction
36	4	Unknown World	scienceFiction
24	5	Inner Sanctum	film noir
22	6	Manasina Maathu	romance
32	7	It Conquered the World	scienceFiction
41	8	Cool World	fantasy
42	9	Monster A Go-Go	scienceFiction

Results for Query 4:

Rank	Re-rank	title	genre
7	0	Dr. Plonk	comedy
31	1	Impostor	scienceFiction
2	2	Akhil	action
47	3	Transcendence	scienceFiction
36	4	Unknown World	scienceFiction
24	5	Inner Sanctum	film noir
22	6	Manasina Maathu	romance
32	7	It Conquered the World	scienceFiction
41	8	Cool World	fantasy
42	9	Monster A Go-Go	scienceFiction

Results for Query 5 :

Rank	Re-rank	title	genre
7	0	Dr. Plonk	comedy
31	1	Impostor	scienceFiction
2	2	Akhil	action
47	3	Transcendence	scienceFiction
36	4	Unknown World	scienceFiction
24	5	Inner Sanctum	film noir
22	6	Manasina Maathu	romance
32	7	It Conquered the World	scienceFiction
41	8	Cool World	fantasy
42	9	Monster A Go-Go	scienceFiction

Query 6:

Rank	Re-rank	title	genre
7	0	Dr. Plonk	comedy
31	1	Impostor	scienceFiction
2	2	Akhil	action
47	3	Transcendence	scienceFiction
36	4	Unknown World	scienceFiction
24	5	Inner Sanctum	film noir
22	6	Manasina Maathu	romance
32	7	It Conquered the World	scienceFiction
41	8	Cool World	fantasy
42	9	Monster A Go-Go	scienceFiction

L. “gpt2-large” results for all queries re-ranked with both models

Re-ranked with Cross-Encoder

Query 1:

Rank	Re-rank	title	genre
16	0	Television Spy	drama
13	1	The Man Who Made Diamonds	crime
24	2	White	romance
8	3	The Green Pastures	fantasy
39	4	Bullets and Saddles	western
44	5	C.H.O.M.P.S.	family
31	6	Nothing Else Matters	comedy
42	7	Wide Boy	crime
28	8	Oyee	romantic comedy
27	9	Key to Harmony	drama

Query 2:

Rank	Re-rank	title	genre
45	0	Love on the Dole	drama
35	1	Red Planet Mars	scienceFiction
10	2	Masked and Anonymous	comedy drama
11	3	Remote Control	comedy
42	4	Sri Naga Shakthi	drama
23	5	Interkosmos	drama
1	6	The Green Pastures	fantasy
16	7	Taurus	biopic
46	8	Rangappa Hogbitna	comedy

19	9	White	romance

Query 3:

Rank	Re-rank	title	genre
30	0	RuPaul Is: Starbooty!	comedy
39	1	Ida Makes a Movie	drama
13	2	Once Upon a Dream	comedy
45	3	Gora Aur Kala	action
49	4	The Kentucky Fried Movie	comedy
42	5	332 Mumbai To India	thriller
3	6	The Green Pastures	fantasy
12	7	Storytelling	drama
1	8	Driverless	romance
16	9	Snow Blind	documentary

Query 4:

Rank	Re-rank	title	genre
25	0	Fleet of Time	romance
6	1	Only for You	romance
42	2	Miss Sadie Thompson	musical
19	3	Once Upon a Dream	comedy
1	4	Driverless	romance
45	5	Ninaithu Ninaithu Parthen	romance
21	6	Min & Max	comedy romance
49	7	Boyss Toh Boyss Hain	comedy
18	8	Key to Harmony	drama
41	9	Faces	drama

Query 5:

Rank	Re-rank	title	genre
47	0	White	romance
44	1	Diggers	comedy
20	2	Dakota Lil	western
49	3	Accused	drama
17	4	Why Don't You Play in Hell?	drama
41	5	The High Powered Rifle	western
35	6	Masterminds	drama
19	7	Loan Shark	film noir
1	8	Sweety Nanna Jodi	romance
22	9	Cocktails	comedy

Query 6:

Rank	Re-rank	title	genre
44	0	Return of Mr. Superman	action
9	1	The Legend of Wisley	adventure
5	2	Good Times	musical comedy
28	3	Hellzapoppin'	musical comedy
8	4	Sri Naga Shakthi	drama
19	5	Rock Dancer	musical
18	6	RuPaul Is: Starbooty!	comedy
31	7	I'm Bout It	drama
47	8	Hawk(e): The Movie	comedy
14	9	Min & Max	comedy romance

Re-ranked with BERTScore

Query 1:

Rank	Re-rank	title	genre
10	0	Ghamandee	action
0	1	Walk Like a Dragon	drama
7	2	Aadesh - Power Of Law	drama
15	3	Hum Farishte Nahin	action thriller
35	4	Kote	action
9	5	Jawaani	romance drama
28	6	Oyee	romantic comedy
6	7	Seesa	horror
31	8	Nothing Else Matters	comedy
48	9	Wind	drama

Query 2:

Rank	Re-rank	title	genre
38	0	A Medal for Benny	drama
14	1	The Beach Party at the Threshold of Hell	comedy
33	2	Sipaayi	drama
1	3	The Green Pastures	fantasy
40	4	Walk Like a Dragon	drama
29	5	Women of the Prehistoric Planet	scienceFiction
24	6	Laali Ki Shaadi Mein Laaddoo Deewana	comedy drama

49	7	The Big Blockade	war
22	8	Manasina Maathu	romance
47	9	Crash and Burn	scienceFiction

Query 3:

Rank	Re-rank	title	genre
33	17648	Walk Like a Dragon	drama
28	21037	Aidondla Aidu	drama
30	13581	RuPaul Is: Starbooty!	comedy
21	18714	Irma Vep	drama
44	5396	Chamber of Horrors	horror
34	13055	Sex and the College Girl	comedy
20	9816	Sweety Nanna Jodi	romance
18	17936	The Last Movie	drama
26	15775	Kal Manja	comedy
14	10332	Listen	thriller

Query 4:

Rank	Re-rank	title	genre
7	2669	Laali Ki Shaadi Mein Laaddoo Deewana	comedy drama
8	9807	Manasina Maathu	romance
24	9809	Murali Meets Meera	romance
13	9596	Hridayer Shabdo	romance
14	20083	Emily Brontë's Wuthering Heights	drama
32	15851	Just Gammat	comedy
2	20592	Love Actually... Sucks!	drama
0	19801	The Woman's Angle	drama

41	17846	Faces	drama
12	9849	White	romance

Query 5:

Rank	Re-rank	title	genre
13	1251	Kodi Veeran	action drama
36	21049	Sipaayi	drama
11	15851	Just Gammat	comedy
14	2669	Laali Ki Shaadi Mein Laaddoo Deewana	comedy drama
37	21550	Heaven's Story	drama
19	4981	Loan Shark	film noir
2	9807	Manasina Maathu	romance
0	6290	Seesa	horror
34	5071	Underworld U.S.A.	film noir
26	1741	Tiger Number One	action romance

Query 6:

Rank	Re-rank	title	genre
13	10037	Graduate	romance
40	739	Ghamandee	action
32	9807	Manasina Maathu	romance
36	2669	Laali Ki Shaadi Mein Laaddoo Deewana	comedy drama
14	2854	Min & Max	comedy romance
1	14847	Nothing Else Matters	comedy
0	9809	Murali Meets Meera	romance
26	5228	I Was a Teenage Zombie	horror comedy
10	21032	Thavarina Runa	drama
22	18546	Wind	drama