

UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

MASTER THESIS IN INGEGNERIA INFORMATICA

Design and Development of an Extensible and Configurable Framework for Conversational Search Experiments

MASTER CANDIDATE

Marco Alessio

Student ID 1242412

SUPERVISOR

Prof. Nicola Ferro

University of Padua

CO-SUPERVISOR

Dott. Guglielmo Faggioli

University of Padua

ACADEMIC YEAR
2022/2023

Abstract

The Conversational Search (CS) paradigm allows for an intuitive interaction between the user and the system through natural language sentences and it is increasingly being adopted in various scenarios. However, its widespread experimentation has led to the birth of a multitude of CS systems with custom implementations and variants of Information Retrieval (IR) models. This exacerbates the reproducibility crisis already observed in several research areas, including IR. To address this issue, we propose DECAF: a modular and extensible Conversational Search framework designed for fast prototyping and development of conversational agents. Our framework integrates all the components that characterize a modern CS system and allows for the seamless integration of Machine Learning (ML) and Large Language Models (LLMs)-based techniques. Furthermore, thanks to its uniform interface, DECAF allows for experiments characterized by a high degree of reproducibility. DECAF contains several state-of-the-art components including query rewriting, search functions under Bag-of-Words (BoW) and dense paradigms, and re-ranking functions. Our framework is tested on two well-known conversational collections: TREC CAsT 2019 and 2020 and the results can be used by future practitioners as baselines. Our contributions include the identification of a series of state-of-the-art components for the CS task and the definition of a modular framework for its implementation.

Contents

List of Figures	xi
List of Tables	xiii
1 Introduction	1
2 Background	5
2.1 Basics of Information Retrieval	5
2.1.1 Documents	7
2.1.2 Information Need	7
2.1.3 Query	7
2.1.4 Analysis	8
2.1.5 Indexing	9
2.1.6 Matching	11
2.1.7 Re-ranking	18
2.1.8 Retrieval Results	19
2.1.9 Evaluation	19
2.2 Conversational Search	24
2.2.1 Query Rewriting	25
2.2.2 Heuristics for Query Generation	26
2.3 TREC Conversational Assistance Track	26
2.4 IR Frameworks	27
3 DECAF Architecture	29
3.1 Main Modules	29
3.2 Components Implemented	30
3.2.1 Index Pipeline	30
3.2.2 Search Pipeline	32

CONTENTS

3.3	External Process Integration	36
3.3.1	Creation	36
3.3.2	Synchronization and Data Exchange	37
3.3.3	Clean-up	38
3.3.4	Techniques to avoid Stream Cluttering in Python	38
3.4	Git Repository	40
4	DECAF in Action	45
4.1	Installation Guide	45
4.1.1	Requirements	45
4.1.2	Dependencies	45
4.1.3	Installation	49
4.1.4	Uninstall	52
4.2	Configuration Guide	52
4.2.1	Index Phase	53
4.2.2	Search Phase	54
5	Experimental Results	57
5.1	TREC CAsT 2019 Results	59
5.1.1	Rewriters and Query Generation Heuristics	59
5.1.2	First-stage Retrieval	66
5.1.3	Re-ranking	67
5.1.4	Comparison with Best Runs of TREC CAsT 2019	70
5.2	TREC CAsT 2020 Results	75
6	Conclusions	81
	References	83
	Acknowledgments	93

List of Figures

2.1	Architecture of IR systems.	6
3.1	Architecture of DECAF.	29
3.2	Typical data and execution flow between the Main and External processes in DECAF.	37
3.3	Screenshot of the main directory of DECAF repository.	41
3.4	Screenshot of the <i>template</i> sub-directory of DECAF repository.	42
4.1	An example of configuration file snippet that allows for configuring the indexing phase.	53
4.2	An example of a configuration file snippet that allows for configuring the search phase.	56
5.1	Evaluation of FLC heuristic paired with three different rewriting strategy. The scoring function is BM25 and no re-ranking is employed. Each column and row represent the coefficient applied to the First and Last utterances, respectively. Instead, the coefficient for Current utterance is given by $1 - (F + L)$. The evaluation metric reported is Normalized Discounted Cumulative Gain (nDCG)@10, considering only utterances those turn is ≥ 3 within its conversation.	62
5.2	Differences in nDCG@10 evaluation metric across all queries, between the run employing FLC w.r.t. considering only the current utterance. Three different rewriting strategies are employed, one for each plot, the scoring function is BM25, and no re-ranking is used. Only the utterances those turn is ≥ 3 within its conversation are considered.	63

LIST OF FIGURES

5.3 Evaluation of the same experiments showed on Figures 5.1 and 5.2 conducted on the “manual” utterances, which were manually rewritten from the organizers of TREC CAsT 2019. 64

5.4 Differences in nDCG@10 evaluation metric across all queries, between the run employing reranking with BERT w.r.t. no reranking. Both experiments are based on Text-To-Text Transfer Transformer (T5)-rewritten automatic utterances, but two different ranking function are employed, one for each plot: BM25 and Dirichlet Language Model (LM). 69

5.5 Evaluation of two different strategies for improving re-ranking. Figure 5.5a evaluates the optimal parameters for the FLC heuristic, when applied to the query representation used for re-ranking with BERT, employed to T5-rewritten automatic utterances. Figure 5.5b shows how the score changes when varying the linear combination parameter in our custom Reciprocal Rank run fusion technique, employed to merge the ranked lists produced by the Searcher (left) and the Reranker (right). The performance measure considered is nDCG@10 in both plots. 71

5.6 Evaluation of the nDCG@10 metric across all queries, for both our best experiment on automatic run, which employs SPLADE, and for the Best Automatic run on TREC CAsT 2019 dataset. 73

5.7 Evaluation of the nDCG@10 metric across all queries, for both our best experiment on manual run, which employs SPLADE, and for the Best Manual run on TREC CAsT 2019 dataset. 74

5.8 Average nDCG@3 metric at varying conversation depths, on TREC CAsT 2019 dataset. In the top plot, there are automatic runs, while the other displays manual runs. The considered runs are: BM25 with BERT (blue bar, on the left), Dirichlet LM with BERT (orange bar, in the center-left), SPLADE (grey bar, in the center-right), and the Best run across original submissions (green bar, on the right). 76

- 5.9 Average nDCG@3 metric at varying conversation depths, on TREC CAsT 2020 dataset. In the top plot, there are automatic runs, while the other displays manual runs. The considered runs are: BM25 with BERT (blue bar, on the left), Dirichlet LM with BERT (orange bar, in the center-left), SPLADE (grey bar, in the center-right), and the Baseline run across original submissions (green bar, on the right). 79

List of Tables

2.1	Example of run in <i>trec-eval</i> format.	20
2.2	Example of relevance judgements in <i>trec-eval</i> format.	20
5.1	Rewriters and corresponding configuration parameters, used in the evaluation of DECAF.	57
5.2	Searchers (first-stage ranking functions) and corresponding configuration parameters, used in the evaluation of DECAF.	58
5.3	Rerankers and corresponding configuration parameters, used in the evaluation of DECAF.	59
5.4	List of experiments conducted on TREC CAsT 2019 benchmark, testing different rewriting strategies and query generation heuristics, detailed in Section 5.1.1. The evaluation measures reported are Recall (R)@100, Mean Reciprocal Rank (MRR), and nDCG with cutoffs 3 and 10.	59
5.5	Optimal parameters to employ with FLC heuristic for maximizing nDCG@10 metric, determined with various rewriting strategies and type of utterances, on TREC CAsT 2019 benchmark.	65
5.6	List of experiments conducted on TREC CAsT 2019 benchmark, testing different ranking functions, detailed in Section 5.1.2. The evaluation measures reported are R@100, MRR and nDCG with cutoffs 3 and 10.	66
5.7	List of experiments conducted on TREC CAsT 2019 benchmark, applying re-ranking using a BERT model to different settings, detailed in Section 5.1.3. The evaluation measures reported are R@100, MRR and nDCG with cutoffs 3 and 10.	68

LIST OF TABLES

5.8	Comparison between our best-performing experiments against the best runs across original submissions, for both automatic and manual utterances, on TREC CAsT 2019 benchmark. The evaluation measures reported are R@100, MRR and nDCG with cutoffs 3 and 10.	72
5.9	List of all experiments conducted on TREC CAsT 2020 benchmark. The evaluation measures reported are R@100, MRR and nDCG with cutoffs 3 and 10. We do not report Recall@100 and nDCG@10 for the best TREC CAsT 2020 runs, since they were not computed in [7].	77



Introduction

Information is an abstract concept that refers to any data or knowledge pertaining to some aspect of the world that are processed, interpreted and organized in such a way to become meaningful for a determined set of people. It can take an heterogeneous and variegated spectrum of representations: written text, such as letters and books, images, like pictures and drawings, video, and audio, including music and speech. Moreover, information can be stored, to retain and make it available for future reference and analysis. Information communication, on the other hand, allows for the distribution and sharing of information across individuals, which is vital for decision-making, learning, and innovation in various domains and disciplines.

Information Retrieval George Salton, a pioneer in IR from the 1960s to the 1990s, proposed the following definition [47]:

Information Retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information.

The demand for IR stems from the lack of knowledge needed to perform a task. To satisfy this information need, the user seeks to find correct and useful information from a collection of resources, delivered to him/her in an appropriate way.

Traditional IR systems are rather limited: users are required to manually translate their information need into an – hopefully effective – query, which is typically a set of keywords. Despite its simplicity, such approach has been

widely employed in practice, ranging from traditional library catalogues to modern search engines. It can be defined as a *one-shot* process: each search is independent from others.

Conversational Search CS [64, 3] is an emerging paradigm which is drastically innovating IR by allowing users to interact with the system through a conversation. This paradigm allows for a very intuitive and seamless interaction between the human and the system since it is based on natural language utterances.

As a result of remarkable technological advancements in recent years, we have seen a significant rise in the adoption of personal assistants equipped with interactive voice-based interfaces, such as *Amazon Alexa*, *Google Assistant*, *Samsung Bixby*, *Apple Siri*, and many others, which testify to the importance of CS in the industry.

Context CS poses significant challenges due to the inherent imprecision and ambiguity of natural language expressions, as well as the presence of complex speech structures in the utterances, including anaphoras, ellipses, and coreferences. Therefore, the system needs to address these natural language phenomena by keeping track of the conversation state and to preserve all the implicit information contained in the utterances.

Neural Approaches In recent years, ML have been successfully employed to perform many Natural Language Processing (NLP) and IR tasks, achieving state-of-the-art performance. The Transformer architecture [50], serving as the foundation of several popular LLMs such as BERT [10] and GPT [40], utilise the self-attention mechanism to capture the relationships between different parts of the input sequence and learn contextual dense representations for each of them.

Additionally, the higher training parallelization w.r.t. previous models have fostered the development of LLMs pretrained on large datasets, which can be fine-tuned for performing specific tasks such as query rewriting [43], first-stage retrieval [55, 66, 15], ranking [19, 54, 61], and question answering [10].

Motivations The advent of Neural Information Retrieval (NIR) models, fueled by LLMs, has been helping to overcome some of these challenges and to foster a capillary adoption of CS in many different scenarios. Such a variety made the CS

scenario fragmented from the systems’ design perspective, with hundreds of ad-hoc custom implementations and variants of IR models and other components developed.

They employ a wide range of languages and frameworks, making their integration difficult, if not impossible at all. This situation determines a number of challenges both in the development of CS systems and in their experimentation. Finally, it hinders the reproducibility of experiments, exacerbating the already prevalent reproducibility crisis in current research [60, 6, 14, 21].

Contributions To address these issues, we propose **DECAF** – *moDular and Extensible Conversational seaArch Framework*, which is explicitly designed to allow for fast prototyping and development of CS systems and to enable their systematic evaluation under the traditional Cranfield paradigm. The framework is designed to allow the integration of all the components that characterize a modern CS system, including query rewriting, searching – both under the BoW and dense paradigms – and re-ranking.

While written primarily in Java to ensure compatibility with traditional IR libraries, such as Lucene¹, Terrier [37], and Anserini [26], it also supports modern Artificial Intelligence (AI), ML, and LLM-based techniques thanks to the seamless integration of Python scripts. The framework already includes multiple state-of-the-art components for query rewriting, traditional BoW similarity functions, NIR approaches – both sparse and dense – and re-ranking functions. It also allows for the evaluation on reference collections in the area, such as TREC CAsT 2019 and 2020.

The components implemented in DECAF act as state-of-the-art baselines for future experiments. They also provide a template for integrating and designing new components, to extend DECAF itself. To show the capabilities of this framework, we demonstrate its application to the TREC CAsT 2019 and 2020 collections, reporting the results that different pipelines are able to achieve.

Our main contributions are the following:

- Design and develop the DECAF modular and easily extensible framework, which allows us to seamlessly integrate CS components and instantiate CS pipelines.

¹<https://lucene.apache.org/>

- Provide a series of state-of-the-art components that can be used to implement a CS pipeline.
- Evaluate several CS pipelines, created using DECAF, on two well-known and widely adopted conversational collections, namely TREC CAsT 2019 and TREC CAsT 2020.

Structure of this work The remainder of this work is organized as follows: Chapter 2 surveys the current state-of-the-art for IR and CS experimentation frameworks. Chapter 3 introduces the framework and its architecture, along with the different CS components implemented out of the box. Chapter 4 analyzes a practical application of our framework, focusing on how to set up the configuration files to conduct experiments. Chapter 5 reports experiments conducted to evaluate the components implemented within DECAF. Finally, Chapter 6 draws some conclusions and outlines future extensions of DECAF.

2

Background

In this chapter, we report the main endeavours and related works concerning four topics: IR and CS fields, the principal evaluation campaigns in the CS setting – used as a reference point to design DECAF, and the related IR frameworks.

2.1 BASICS OF INFORMATION RETRIEVAL

The practice of archiving and retrieving information is a fundamental activity of human society. In ancient times, this was accomplished through oral tradition, involving the memorization and recitation of stories and myths. The development of writing led to the creation of various methods for organizing and cataloging written works. In the latter half of the 20th century, the technological advancement have fueled the development of automatic systems for storing and retrieving information. The term Information Retrieval has been firstly introduced by Calvin Mooers in 1950 [13], denoting the recovery, from a given collection of documents, of a subset including all documents of specified content, possibly together with irrelevant ones.

From there, a lot of research developed. IR systems have converged towards a standard architecture, shown in Figure 2.1. It involves an offline phase, that handles preprocessing of the entire collection of information, and an online phase, that is activated for each query to process. This architecture has been widely adopted and has proven to be effective for IR systems.

2.1. BASICS OF INFORMATION RETRIEVAL

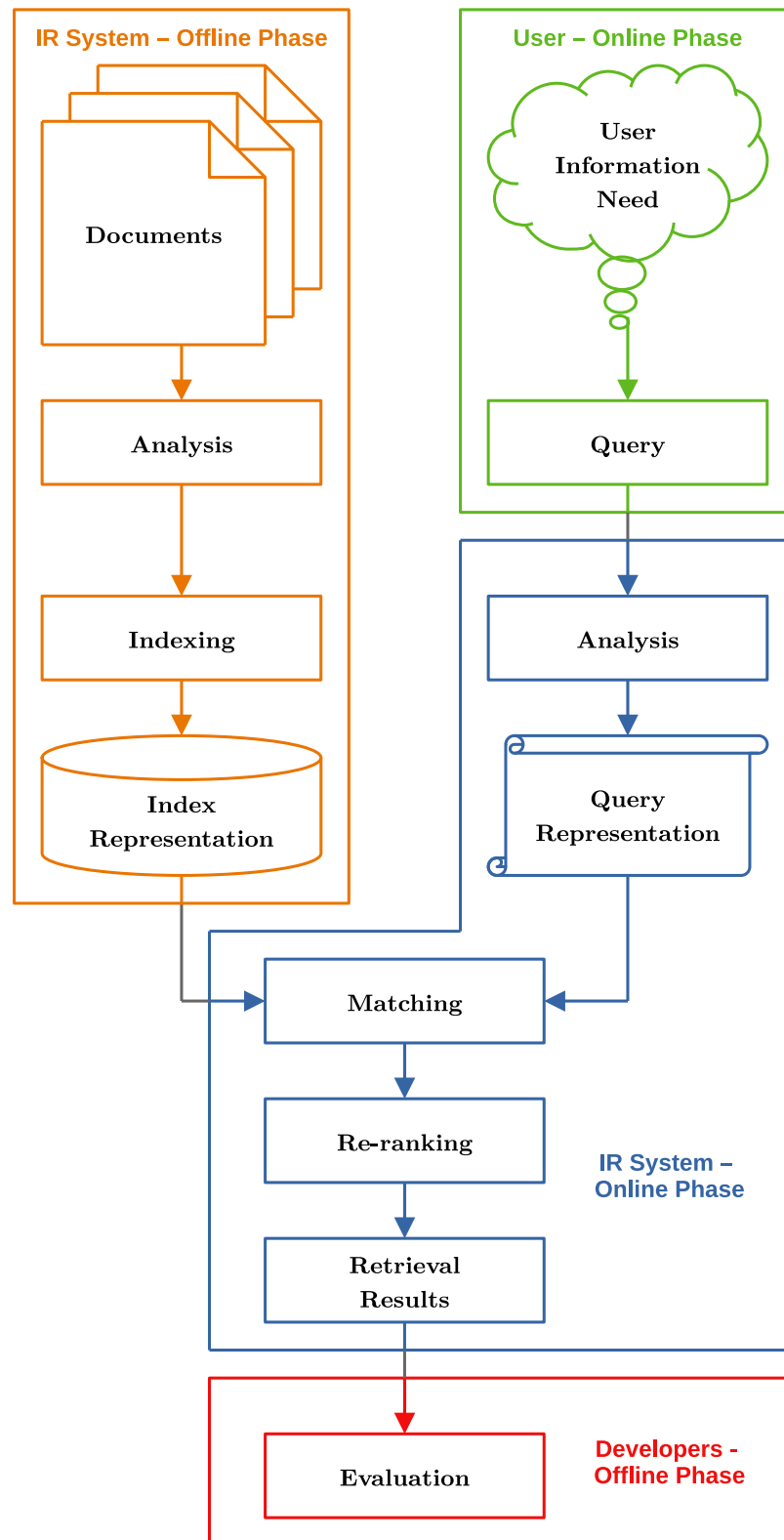


Figure 2.1: Architecture of IR systems.

2.1.1 DOCUMENTS

IR systems operate on a collection of resources, which represent the entire pool of information available to the system. Historically, emphasis have been placed on textual-based content including books, papers, email, and web pages. However, modern applications of IR increasingly encompass multimedia resources, such as pictures, videos, and audio.

In this work, we consider exclusively textual documents. While they may possess some degree of structure, such as title, author, chapters and paragraphs, most of the information is conveyed through unstructured text.

2.1.2 INFORMATION NEED

The information need refers to the demand for accurate and useful knowledge required by users of IR systems to perform a specific task. Based on the different users' requirements and expectations, we can distinguish four kinds of IR systems:

Information Seeking The user wishes to locate information relevant to the provided topic.

Question Answering These systems are concerned with delivering automatically-generated answers to questions posed in natural language by the human users.

Recommendation Systems The aim is to help users providing a set of items, often taken from a closed domain, obtained by filtering and selecting those that are considered the most suitable to fulfill the user's needs.

Goal-oriented Systems They are designed to aid people in fulfilling some specific tasks, such as making a reservation, or buying a product.

2.1.3 QUERY

The user translates his/her information need into the query, usually involving a set of keywords that are deemed the most appropriate terms for the system to retrieve the desired data to fulfill the information need.

2.1. BASICS OF INFORMATION RETRIEVAL

The effectiveness of retrieval is somewhat dependent on the specific formulation chosen by the user, since different but equivalent-meaning queries may produce non-identical results.

2.1.4 ANALYSIS

The analysis phase is conducted independently for each document and query, with the objective of extracting the features that describe their content. The specific activities carried out during this phase depend on the particular indexing approach and retrieval model employed in the pipeline. The most common sub-phases are tokenization, stopword removal, and stemming.

TOKENIZATION

Tokenization is always the first step in the analysis process performed by IR systems. The textual content is transformed into a stream of tokens, which represent the minimal lexical units considered by the system, and typically correspond to individual words. Punctuation marks, special characters, and blank spaces are used as delimiters to separate words from each other. There are special cases that must be treated with care. Hyphenated words sometimes should be handled as a single word (i.e., ice-cream, co-operation), while in other cases as separate words (i.e., user-centered, dog-friendly, well-known). Apostrophes could be a part of a word or of a possessive, or just a mistake. Uppercase letters are usually insignificant, but there are exceptions: for instance, "Bank" is a city in Iran and "bank" is a financial institution.

Finally, all words are converted to lowercase and punctuation marks are discarded.

STOPWORD REMOVAL

Stopword Removal is an optional step that involves filtering out stopwords, which are terms with low discriminatory power for retrieval purposes. The underlying principle is that words occurring in the majority of the documents, such as articles, prepositions, conjunctions, and pronouns, are not informative and can be safely discarded. The primary benefit of this technique is the reduction in the size of the index, whose extent depends on the aggressiveness of the elimination process. However, a potential drawback is the loss of important information embedded within the excluded words.

STEMMING

Stemming is an optional process which attempts to reduce morphological variations, such as plural, gerund, and past tense forms, of related words by eliminating common prefixes and suffixes. The portion of the word remaining after this process is referred to as the "stem".

The usage of stemming offers certain advantages: most words pertaining to a common concept are reduced into the same term, while coincidentally reducing the number of unique words and, in turn, the size required by the index structure. On the other hand, the stemming algorithm may occasionally produce wrong results lowering the effectiveness of this technique. Overstemming is when some unrelated words are transformed into the same term (i.e., "university" and "universe"), while understemming is when related words are reduced to different stems (i.e., "mouse" and "mice").

In the literature, there is controversy surrounding the usage of stemming for english language, with conflicting conclusions being drawn in different research works. However, stemming is de-facto mandatory in other languages, such as arabic.

2.1.5 INDEXING

The corpus of documents is processed ahead of time in order to build an index, a data structure needed to efficiently support the retrieval of documents in response to the user query. The purpose of this step is to avoid performing a linear scan of the entire documents collection, resulting in faster response times. Information Retrieval systems employ a particular type of search called ranking, which involves retrieving documents in sorted order based on a score that is computed by applying a scoring function to the query and documents representations.

BAG-OF-WORDS REPRESENTATIONS

While performing indexing, it is possible to build the dictionary, a list containing all distinct terms that appeared at least once in the documents corpora. BoW are representations in which each term is associated with a weight, reflecting the relative importance of the considered word in the document. The peculiar aspect of BoW representations consists in the fact that they are sparse: only a

2.1. BASICS OF INFORMATION RETRIEVAL

small number of terms are assigned to non-zero weights. Usually, the BoW representation for a document is composed by the set of unique words appearing in its text associated with their frequency.

The inverted index is the most popular data structure employed with Bag-of-Words representations, providing a mapping between terms and the documents in which they appear. The index is referred to as “inverted”, because starting from documents composed of terms, this data structure invert such relation by associating each term with the corresponding documents.

Every document in the collection is linked with an unique number. Each term is associated with its own posting list: each entry contains the unique identifier of a document where the term appears, optionally together with other features containing useful information used to perform retrieval. Common examples of such features are term frequency and positions list, allowing for ranking documents by their relevance as well as for phrase and proximity matching.

Inverted lists are usually stored together in a single file for efficiency purposes. There are auxiliary data structures for other useful data: vocabulary, term and collection statistics are commonly stored in separate files.

DENSE REPRESENTATIONS

The remarkable modeling capacity of pretrained LLMs allowed to effectively learn the representations for textual data in the latent representation space. The latent space is an embedding of a collection of items, where entities with similar characteristics are positioned closer to each other in the latent space. The embedding refers to the output of an injective function $f: X \mapsto Y$, in which each item belonging to X is mapped into a different entity, which in our case is a vector of real numbers.

These vectors, belonging to the same high-dimensional vector space, can be compared using a semantic matching function, such as dot product and cosine similarity. The score obtained is a measure of similarity between the actual texts being compared, as guaranteed by the latent space.

One widely adopted pretrained LLM is BERT [10], which generates a contextual representation for each token found in the text. They belong to a vector space of size 768 or 1024, depending on which variant (“BERT_{BASE}” or “BERT_{LARGE}”) is employed. During tokenization phase, the special token [CLS] is added at the beginning, which is important as its embedding encapsulates the

meaning of the whole text. For this reason, only this embedding is retained; this approach is known as the CLS pooling technique.

Dense retrieval approaches commonly employ a list as index structure, comprising the embeddings of CLS token that are generated for each document. These indexes require to store a substantial amount of data. For example, if a BERT_{BASE}-based model is utilized, each document require $768 \cdot 4 \text{ bytes} = 3 \text{ KB}$ of storage space. This can become problematic when indexing large corpora of documents. For instance, the corpora used in the TREC CAsT 2019 and 2020 evaluation campaigns consist of approximately 38.5 million documents, therefore necessitating a total storage space of 110.15 GB. During retrieval stage, the query embedding is compared to all document embeddings to identify the top- k most relevant documents. This brute-force approach raises efficiency concerns regarding computational power, memory, and time required.

In order to address these issues, various approximate search techniques have been developed. There are approaches in which the embeddings are approximated, in order to decrease their storage requirements. Other approaches aim to disregard the majority of documents during retrieval, as they are deemed unlikely to be among the top- k most relevant documents for the given query. Clustering is commonly employed to partition the embeddings into smaller subsets, those centroids are used to determine which partitions are examined and which are ignored. These approximate search techniques have been demonstrated to be effective, although their effect on performance metrics has not been fully explored and remains an area of active research.

2.1.6 MATCHING

This phase is concerned with determining which documents the user will find relevant to fulfill his/her information need. The formalization of the processes undertaken by a person deciding about the relevance of a piece of text has been one of the primary goals of research in the IR field.

Retrieval models establish a mathematic framework to define the search process, based on certain assumptions. They serve as the foundation for many ranking functions that compute a score for each document, representing the confidence about its relevance for the given query. Effective models produce outputs that highly correlates with human judgements on relevance.

Over the course of the years, several retrieval models have been proposed.

2.1. BASICS OF INFORMATION RETRIEVAL

The main ones are the Boolean Model, the Vector Space Model and the Probabilistic Model.

BOOLEAN MODEL

The Boolean is a classical retrieval model that has been the first and most-adopted model by early IR systems. Despite its simplicity, it is still employed in some specific domains, such as digital libraries and legal search. The Boolean model is based on the principles of boolean algebra and set theory. Each document is represented as a set of terms, which are typically single words or phrases present in the text. Both the frequency of terms and the order in which they appear are not considered in this model. Instead, queries are formulated as boolean expressions composed of terms linked by logical operators such as NOT, AND, OR, and XOR.

The ranking function in this model is a binary match: the documents are either relevant or not, with no partial matching allowed. The result does not consist in a ranked list, but rather a set of documents. For this reason, the Boolean model can be defined as an exact-match model, as documents are retrieved only if they exactly match the query specification.

This retrieval model offers several advantages. It is particularly suited for exact match queries, and provides predictable and easily explainable results. Furthermore, efficient processing is achievable since most documents can be ignored during the search process.

On the other hand, it comes with many drawbacks. Firstly, a common outcome results in retrieving too few or too many documents. For this reason, both simple and complex queries are seldom effective. Secondly, the model effectiveness depends entirely on user's ability to express his/her information need through a well-designed logical expression. Thirdly, it does not account for term frequency and order, which doesn't allow to search for specific phrases as well as using proximity operators. Lastly, the model does not provide any recommendation on which documents the user should examine first.

VECTOR SPACE MODEL

The Vector Space Model is a framework for implementing term weighting, ranking and relevance feedback, proposed during the 1960s.

In this model, both the documents and queries are represented as vectors in a

N -dimensional vector space. The similarity between a query and a document is evaluated using either the dot product or cosine similarity between their vector representations. Therefore, this model allows for partial matching and ranked results.

Dot Product

$$rel(q, d) = \vec{q} \cdot \vec{d} = \sum_{i=1}^N q_i \cdot d_i$$

Cosine Similarity

$$rel(q, d) = \cos(\theta(\vec{q}, \vec{d})) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{\sum_{i=1}^N q_i \cdot d_i}{\sqrt{\sum_{i=1}^N q_i^2} \sqrt{\sum_{i=1}^N d_i^2}}$$

Despite the absence of an explicit definition of relevance in this model, there exists an underlying assumption that relevance is related to the similarity between the vectors representing the query and the document. Instead, no assumptions are made regarding the nature of relevance, whether it is binary or multi-graded.

The representation of documents and queries is determined by the specific ranking algorithm being utilized.

For instance, classical approaches based on the Vector Space Model consider a dictionary of N possible terms, and associate each of them with a weight characterising the term importance for the given entity. The vector representation consists in the aggregation of all the weights assigned to the terms. The rationale behind it is based on the premise that not all terms are equally useful for describing the content. For example, common words such as articles and conjunctions are not informative; on the other hand, a rare term appearing in only a few documents have a strong discriminatory power, allowing to considerably narrow down the set of potential relevant documents during retrieval. The weights are typically derived from the frequency of the terms in the corpus of documents.

The TF-IDF (Term Frequency-Inverse Document Frequency) weighting scheme is composed of two factors: the TF component measures terms occurring with substantial frequencies within a document/query, while the IDF component evaluates the specificity of a term, i.e., how well it characterizes the topic de-

2.1. BASICS OF INFORMATION RETRIEVAL

scribed in the text.

$$w_{t,d} = tf(t, d) \cdot idf(t) = \begin{cases} (1 + \log_2(f_{t,d})) \cdot \log_2 \frac{N}{n_t} & \text{if } f_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where t and d are, respectively, the term and document considered, $f_{t,d}$ is the frequency of term t within the document d , and n_t is the number of documents in which the term appears.

This retrieval model offers several advantages. Firstly, it is a simple, efficient and general computational framework for ranking. Secondly, it allows for partial matching of query terms, therefore avoiding the problem affecting the Boolean retrieval model of manually finding the right trade-off between coverage and specificity of retrieval. Lastly, it returns a list of documents, ranked by their predicted relevance w.r.t. the information need of the user.

The primary drawback is the heuristic approach adopted by this retrieval model, which is based on basic mathematical operations and lacks a robust theoretical foundation.

PROBABILISTIC MODEL

The Probabilistic retrieval model is supported by a robust foundation for representing and manipulating the uncertainty which is an inherent aspect of the IR process. Assuming that the relevance of a document is independent of other documents, it can be shown that ranking documents by their probability of relevance leads to optimal performance of the IR system.

For any ranking algorithm which assumes binary relevance, it is possible to partition documents into two distinct categories, namely relevant and non-relevant. Given a query, the primary objective of IR systems is to determine whether a document belongs either in the relevant or non-relevant set. Thus, the problem can be characterized as a classification task.

To accomplish this, the probabilities of relevance, $\mathbb{P}(R \mid d, q)$, or non-relevance, $\mathbb{P}(\bar{R} \mid d, q)$, given a document representation are evaluated, and the document is classified into the category with the highest probability. By applying the Bayes theorem, which states that $\mathbb{P}(A \mid B) = \mathbb{P}(B \mid A) \cdot \mathbb{P}(A) / \mathbb{P}(B)$, the decision rule can be expressed as:

$$d \in D_{relevant} \leftrightarrow \mathbb{P}(R | d, q) > \mathbb{P}(\bar{R} | d, q) = \frac{\mathbb{P}(d | R, q)}{\mathbb{P}(d | \bar{R}, q)} > \frac{\mathbb{P}(\bar{R}, q)}{\mathbb{P}(R, q)}$$

The right-end side of the inequality is independent from the document, and can be ignored for the ranking task when considering the left-hand side (known as likelihood ratio) as a score. Therefore, it is sufficient to estimate $\mathbb{P}(d | R, q)$ and $\mathbb{P}(d | \bar{R}, q)$ probabilities.

BM25 The BM25 scoring function has been proved to exhibit high performance in various TREC evaluation campaigns. For this reason, it is widely employed as a baseline in the majority of research endeavors within the IR field.

BM25 is based on the Binary Independence Model (BIM), which does three assumptions:

- Each document is represented by a vector on N binary random variables indicating term occurrence or non-occurrence.
- Terms are statistically independent when considering relevance.
- Only the terms appearing in the query are considered for relevance of documents. Terms not appearing in the query have the same probability of occurrence in the relevant and non-relevant documents, so they are negligible for ranking.

Despite not being realistic assumptions, they enable a substantial simplification of the problem. The probabilities $\mathbb{P}(d | R, q)$ and $\mathbb{P}(d | \bar{R}, q)$ can be estimated as follows:

$$\mathbb{P}(d | R, q) = \prod_{i \in Q} \mathbb{P}(d_i | R) \qquad \mathbb{P}(d | \bar{R}, q) = \prod_{i \in Q} \mathbb{P}(d_i | \bar{R})$$

The BM25 scoring function can be derived starting from the foundation provided by the Binary Independence Model. The BIM has been extended to also consider the document terms frequency, based on probabilistic arguments and experimental validation but without theoretical justification. The score is computed by summing, for each query term, the value computed as the product between a *idf*-like and a *tf*-like factors. There exists many variations of this scoring function, but the most common formulation is:

$$\begin{aligned}
BM25(d, q) &= \sum_{q_i \in q} idf(q_i) \cdot tf(d, q_i) \\
&= \sum_{q_i \in q} \log_2 \left(1 + \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} \right) \cdot \frac{f(q_i, d) \cdot (k_1 + 1)}{f(q_i, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avg_len} \right)}
\end{aligned}$$

where N is the number of documents, $n(q_i)$ is the number of documents containing the i -th term of the query, $f(q_i, d)$ is the frequency of term q_i in document d , $|d|$ is the length of document d , and avg_len is the average length of documents. Instead, k_1 and b are parameters whose value is determined empirically. The former determines how the term-frequency component changes as the frequency of term q_i increases for the considered document. If $k_1 = 0$, only the presence or absence of the term is considered. Otherwise, the tf component will tend to saturate to k_1 as the frequency of the term increases, similarly to the usage of $\log_2(f(q_i, d))$ in the TF-IDF weighting scheme. The parameter b regulates the impact of length normalization, given by the denominator of the tf component, which is necessary to fairly evaluate documents with different length. It takes a value between 0 and 1: $b = 0$ disable length normalization, while $b = 1$ is full normalization. Many TREC experiments showed that effective values for these parameters are $k_1 = 1.2$ and $b = 0.75$.

Language Model LMs are commonly utilized in various NLP tasks to represent text. The simplest form of LM, known as unigram, constitutes a probability distribution over the vocabulary of the language, where each word is associated with its probability of occurrence. For search applications, LMs are employed to represent the topical content of a text. While the majority of words within the vocabulary are expected to have low probabilities, those that are important to the considered topic have substantially higher probabilities.

Language Models can generate novel text by sampling words according to its probability distribution. In the query likelihood retrieval model, documents are ranked based on the probability $\mathbb{P}(q | d)$ that the query could be generated from the LM representing the document. The objective is to measure the degree of similarity between the topic of query and document. Such probability is calculated as the product of the probabilities of generating each word independently from the document LM.

$$\mathbb{P}(q \mid d) = \prod_{q_i \in q} \mathbb{P}(q_i \mid d)$$

The probability distribution of the LM is inferred using the maximum likelihood estimation: for each word, the probability is given by the ratio between its frequency and the length of the document considered.

$$\mathbb{P}(q_i \mid d) = \frac{f(q_i, d)}{|d|}$$

However, this estimation will assign a probability of 0 for all query terms not found in the document. Therefore, from the document LM is not possible to generate the query text if it contains unseen word(s), and the score assigned to the considered document is also 0. This behavior is not appropriate: for instance, documents missing one query term should be valued higher than those missing the whole query text.

This phenomena is caused by the small amount of document text available for creating the LM. Smoothing is a technique used to overcome such data sparsity problem: it consists in assigning some non-zero probability to the estimates of words not appearing in the document, usually based on their frequency in the whole documents corpus D . In practice, this is often achieved through a linear combination between the probabilities estimated from the document and the entire documents corpus.

$$\begin{aligned} \mathbb{P}(q_i \mid d) &= (1 - \alpha) \cdot \mathbb{P}(q_i \mid d) + \alpha \cdot \mathbb{P}(q_i \mid D) \\ &= (1 - \alpha) \cdot \frac{f(q_i, d)}{|d|} + \alpha \cdot \frac{f(q_i, D)}{|D|} \end{aligned}$$

There exists many different approaches for specifying the value of α . The Dirichlet is an effective smoothing technique that is dependent on document length.

$$\alpha = \frac{\mu}{|d| + \mu}$$

The value of the μ parameter is set empirically: typical values range from 1000 to 2000. Smaller values boost the importance given to terms frequency, while larger values favor the number of matching terms. If no text is available,

2.1. BASICS OF INFORMATION RETRIEVAL

the term probability would be $\frac{f(q_i, D)}{|D|}$ which is a reasonable estimation based on the corpus data. As more text is observed, the less influence prior knowledge will have.

The expression for α can be plugged in the general smoothing formula, obtaining the following scoring function:

$$\begin{aligned}\mathbb{P}(q_i | d) &= \frac{|d|}{|d| + \mu} \cdot \frac{f(q_i, d)}{|d|} + \frac{|\mu|}{|d| + \mu} \cdot \frac{f(q_i, D)}{|D|} \\ &= \frac{f(q_i, d) + \mu \frac{f(q_i, D)}{|D|}}{|d| + \mu}\end{aligned}$$

SPLADE

SPLADE [15, 16] is a BoW neural retrieval model developed specifically to be used as a first-stage retrieval model, without requiring additional re-ranking to achieve state-of-the-art performance. The particular SPLADE instance employed in our experiments has been fine-tuned for passage retrieval, utilizing two distinct models for documents and queries. It performs document expansion by adding a large number of terms not found in the original text while producing sparser representations for queries.

SPLADE predicts term importance based on the logits of the Masked Language Model (MLM) layer given by a BERT model. Therefore, a real-value weight is generated by the model for each term of the vocabulary. Regularization techniques are employed to learn sparse representations, in which only a small fraction of terms are assigned to non-zero weights, resulting in efficient indexes.

2.1.7 RE-RANKING

Re-ranking is an optional phase which aims to increase the overall effectiveness at small cutoffs by reordering the documents selected during the first-stage retrieval. Since the introduction of LLMs, this step is usually performed using such models to evaluate the semantic similarity between the documents' text and the provided query. The initial retrieval process typically employs scoring functions that focus on lexical features, such as BM25.

RUN FUSION

Run Fusion techniques allows to merge two or more ranked lists generated by different systems, with the objective of producing a single one resulting in higher effectiveness than any of the original lists. In DECAF, we implemented two different methods: linear combination and our custom Reciprocal Rank-based method.

Linear Combination Given two ranked lists A and B , their scores are normalized in the $[0.0, 1.0]$ range by applying min-max normalization. Then, the final score assigned to each document is given by the linear combination between the normalized scores from both lists.

$$s(d) = \alpha \cdot \frac{s_A(d) - \min_A}{\max_A - \min_A} + (1 - \alpha) \cdot \frac{s_B(d) - \min_B}{\max_B - \min_B}$$

Reciprocal Rank For each document, a score based solely on its rank is obtained from both ranked lists A and B , which are then combined using a linear combination. To obtain the same results as the standard Reciprocal Rank run fusion technique [5], use $k = 60$ and $\alpha = 0.5$.

$$s(d) = \alpha \cdot \frac{1.0}{k + \text{rank}_A(d)} + (1 - \alpha) \cdot \frac{1.0}{k + \text{rank}_B(d)}$$

2.1.8 RETRIEVAL RESULTS

In the retrieval phase, a ranked list of retrieved documents with their respective scores is generated. IR experiments typically involve running the system on multiple queries. The rankings obtained are stored in a textual file, referred to as the run, which is used to assess the system's performance measurements. The runs are commonly distributed in the *trec-eval* format, which consists in a tab-separated file with six columns: the ID of the query, a $Q0$, the ID of the document, the rank and the score achieved by the judged document, and the ID of the run. A valid example is shown in Table 2.1.

2.1.9 EVALUATION

Evaluation is an essential phase in any field, enabling to make progress and devise better solutions/products/systems. In the context of IR, evaluation mea-

Table 2.1: Example of run in *trec-eval* format.

Query ID	Fixed	Document ID	Rank	Score	Run ID
32_1	Q0	MARCO_2861203	1	12.734488	Example
32_1	Q0	MARCO_8685439	2	12.662704	Example
32_1	Q0	MARCO_3878347	3	12.305318	Example
32_1	Q0	MARCO_1361406	4	12.227960	Example
32_1	Q0	MARCO_4978407	5	12.056210	Example
32_1	Q0	MARCO_7208611	6	12.044337	Example
32_1	Q0	MARCO_4181532	7	11.873714	Example
32_1	Q0	MARCO_2925873	8	11.605083	Example
32_1	Q0	MARCO_6584633	9	11.598649	Example
32_1	Q0	MARCO_1905581	10	11.403030	Example

Table 2.2: Example of relevance judgements in *trec-eval* format.

Topic ID	Fixed	Document ID	Judgement
32_1	Q0	MARCO_1361406	1
32_1	Q0	MARCO_2322023	2
32_1	Q0	MARCO_2861203	1
32_1	Q0	MARCO_3232784	0
32_1	Q0	MARCO_3955620	0
32_1	Q0	MARCO_4181532	1
32_1	Q0	MARCO_4978407	2
32_1	Q0	MARCO_6584633	2
32_1	Q0	MARCO_8441724	1
32_1	Q0	MARCO_8685439	0

sures how well a particular system performs and enables to determine which is the best performing one in a specific setting.

One of the primary distinctions made in the evaluation is between effectiveness and efficiency. The former can be defined as a measure of the correspondence between the ranking list produced by the IR system and the user relevance judgments, which are based on the subjective perception that each considered document contains relevant information to fulfill his/her information need. The latter, instead, is defined as the amount of resources (such as time, energy, memory, computing power, ...) required by the system to produce the intended result. While important, in this work we focus exclusively on effectiveness, as an

extremely fast system is pointless unless it produces good results.

A fundamental requirement for evaluation consist in ensuring the repeatability of the experiments and a fair comparison between different systems. To achieve this, the experimental data and settings must be fixed.

As a result of the Cranfield projects, a series of early experimentations conducted by Cyril W. Cleverdon during 1950s and 1960s, the Cranfield paradigm has emerged as the standard for effectiveness evaluation in the IR field. It is based on experimental collections, which are composed of documents corpora, topics and relevance judgements.

The set of documents should be representative of the realistic use-case scenario considered in terms of number, size and type of documents. Typically, more than one corpus is employed, ensuring that the results obtained may be generalized and are not corpus-specific.

The topics represent a surrogate for information needs of an hypothetical user. They may either be acquired from query logs of real IR systems or be manually generated by the curators of the collection. Commonly, at least 50 topics are employed for evaluation.

The relevance judgments are usually manually created, requiring a considerable investment of manual effort for the curators as well as representing the most time-consuming step when producing new experimental collections. The judgements are incomplete, as the complete coverage of the whole pool of documents is not feasible. In practice, a technique called pooling is used. It reduces the number of documents that are examined by considering only those appearing in the top-k results collected from multiple systems. All documents left unjudged should be considered as not relevant.

The relevance judgements could be either binary or multi-graded. In the former case, a document is assessed as either relevant or not, while in the latter multiple level of relevance are considered (i.e., not relevant, partially relevant, fully relevant).

The relevance judgements are usually distributed in *trec-eval* format, which consists in a tab-separated file with four columns: the ID of the topic, a Q_0 , the ID of the judged document, and the relevance score assigned to it. A valid example is shown in Table 2.2.

Despite DECAF not including any component to carry out evaluation, we perform this phase using *trec-eval* tool. It is an open-source C program developed by NIST, and it is the standard tool used for every evaluation campaign

2.1. BASICS OF INFORMATION RETRIEVAL

organized within TREC conference. It is available at: https://github.com/usnistgov/trec_eval. The evaluation measures considered in this work are three: R, MRR and nDCG.

For explanation purposes, we have to consider the following quantities. Let D be the set of documents, Q be the set of queries, and $D_k(q)$ be the set of top- k most relevant documents retrieved for a query $q \in Q$. The cutoff, k , is the number of results considered when computing the evaluation metric. Moreover, with $rel(d, q)$ we indicate the relevance assigned for a query $q \in Q$ to document $d \in D$, if available, otherwise 0. A document is considered relevant for a given query if its relevance is greater or equal to a given threshold, rel_{min} , which is usually set to 1. Let $D_{rel}(q) = \{d \in D \mid rel(d, q) \geq rel_{min}\}$ be the set of relevant documents for the query $q \in Q$. Finally, $rank(d, q)$ is the rank given to document $d \in D$ and $d_n(q)$ is the document ranked in position n for a query $q \in Q$.

Recall The Recall is a set-based retrieval measure, evaluating the proportion of relevant documents actually retrieved for a given query. This measure is given by the following formula:

$$R@k(q) = \frac{|D_{rel}(q) \cap D_k(q)|}{|D_{rel}(q)|}$$

It is common practice to report the average value across the set of queries Q .

$$R@k = \frac{\sum_{n=1}^{|Q|} R@k(q_n)}{|Q|}$$

Mean Reciprocal Rank The Reciprocal Rank (RR) is defined as the reciprocal of the rank at which the first relevant document is retrieved for the given query $q \in Q$, otherwise 0 if no relevant documents are retrieved. The Mean Reciprocal Rank is the average of RR over the set of queries Q .

$$RR@k(q) = \begin{cases} \frac{1}{\min\{rank(d,q) \mid rel(d,q) \geq rel_{min} \forall d \in D_k(q)\}} & \text{if } |D_{rel}(q) \cap D_k(q)| \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$MRR@k = \frac{\sum_{n=1}^{|Q|} RR@k(q_n)}{|Q|}$$

Normalized Discounted Cumulative Gain The nDCG is a popular measure for evaluation, based on the assumption that highly relevant documents are more useful than marginally relevant ones. Moreover, the probability that a document is examined by the user decreases as it is positioned further down in the ranked list of results.

Based on these considerations, the nDCG measures the sum of the usefulness (measured by the relevance) gained by the user when examining the ranked list of documents retrieved by the system. The sum is discontinued, as relevant documents appearing lower in the result list are penalized by a weight proportional to the logarithm of its rank. To facilitate the comparison across different cut-offs and queries, the sum (DCG) is normalized in $[0, 1]$ range by dividing with the ideal value obtained when considering the perfect ranking (iDCG), in which documents are sorted by relevance in descending order.

$$DCG@k(q) = \sum_{n=1}^k \frac{rel(d_n(q), q)}{\log_2(1+n)}$$

$$nDCG@k(q) = \frac{DCG@k(q)}{iDCG@k(q)}$$

As with Recall, the average value across the set of queries Q is usually reported.

$$nDCG@k = \frac{\sum_{n=1}^{|Q|} nDCG@k(q_n)}{|Q|}$$

Considering the data shown in Tables 2.1 and 2.2, we can compute the following results:

Recall

$$|D_{rel}| = 7 \qquad |D_{rel} \cap D_{10}| = 5$$

$$R@10 = \frac{|D_{rel} \cap D_{10}|}{|D_{rel}|} = \frac{5}{7} = 0.7143$$

Reciprocal Rank, with $rel_{min} = 2$

$$RR@10 = \frac{1}{\min\{5, 9\}} = \frac{1}{5} = 0.2000$$

2.2. CONVERSATIONAL SEARCH

nDCG@5

$$DCG@5 = 1 * \log_2(1 + 1) + 0 * \log_2(1 + 2) + 0 * \log_2(1 + 3) + \\ 1 * \log_2(1 + 4) + 2 * \log_2(1 + 5) = 8.4919$$

$$iDCG@5 = 2 * \log_2(1 + 1) + 2 * \log_2(1 + 2) + 2 * \log_2(1 + 3) + \\ 1 * \log_2(1 + 4) + 1 * \log_2(1 + 5) = 14.0768$$

$$nDCG@5 = \frac{DCG@5}{iDCG@5} = 0.6033$$

2.2 CONVERSATIONAL SEARCH

Conversational Search (CS) consists of the exchange of natural language utterances between a user and a conversational agent. The most peculiar aspect of this scenario is that the system needs to keep track of the context [41] as the dialogue unfolds. In fact, natural language expressions are inherently imprecise and ambiguous. Complex speech structures in the utterances, such as anaphoras, ellipsis, and coreferences, exacerbate these problems. Anaphora is the repetition of words or phrases at the beginning of a group of successive sentences, to emphasize a particular idea or topic throughout the conversation. Ellipsis refers to the omission of one or more words from a sentence that can be inferred from the context provided by the remaining elements. Coreference occurs when two or more expressions within a text refer to the same entity, hence they have the same referent. One of the most common source of coreference is through the use of pronouns, linguistic elements that substitute for previously defined entities within a discourse. Moreover, users may refer implicitly to entities and topics previously mentioned in the conversation, ask for more details and clarifications, or change the current topic, thus drifting the trajectory of the exchange [2, 31].

Depending on the task they absolve, conversational agents are commonly divided into chit-chat bots [63, 56] and task-oriented systems [18, 4, 38]. The former class of systems is meant to entertain the users, while the latter allows helping the user to achieve a certain goal, such as buying or learning new information, by the means of a dialogue. A further categorization of task-driven CS systems, consists in dividing them into approaches used to retrieve the response within a corpus [53, 48, 18] and systems that construct the response by

combining multiple retrieved sources using summarization approaches such as T5 [43].

The multi-turn conversational task is characterized by the importance given to the “context” [23, 32, 49, 44]. It consists in the system’s internal belief concerning the conversation state while it evolves through time. To keep track of the context, a large part of the research work has been focused on rewriting utterances, enriching them with the correct contextual information provided by the user in previous utterances. In this way, the rewritten utterances become self-explanatory and thus suitable for an IR system [57, 51, 28, 31, 24]. Another approach to modelling the context consists of adopting approaches based on dense retrieval models. For example [62], the teacher-student framework has been employed to learn a student query encoder to use in conjunction with a standard ad hoc dense retrieval teacher model, such as ANCE [55], or TCT-ColBERT [27], in the role of the documents encoder. The student query encoder is trained to replicate the embeddings given by the teacher model for the oracle reformulated queries. This method allows the elimination of the explicit query rewriting phase from the pipeline [62].

2.2.1 QUERY REWRITING

This is an additional phase performed by CS systems before the query analysis phase, with the objective of rewriting the current utterance by enriching it using the correct contextual information gathered and made available throughout the conversation.

Coreference Resolution Co-reference Resolution (CR) aims to determine which are the spans of text referring to the same entity, and replace all of them with the most appropriate text appearing in a occurrence, which in most cases is the first one. The rationale is that all coreferential expressions are not equivalent: the first occurrence is often a full or descriptive form, while later occurrences use shorter forms. However, sometimes pronouns are employed to refer forward to not-already mentioned entities.

In our work, we employ two different CR models each based on a specific framework. The first one is the “coref-spanbert-large” model of AllenNLP library [17], while the other is the “f-coref” model [35] based on the LingMess architecture [36]. Both solutions achieve state-of-the-art performance on the CR

2.3. TREC CONVERSATIONAL ASSISTANCE TRACK

task, although the first one is based on the AllenNLP library which has been deprecated and is no longer updated by the original authors since December 2022.

T5 Model-based The Text-To-Text Transfer Transformer model is a pretrained LLM developed by Google in 2019 [42] based on the Transformers architecture [50]. The peculiar aspect of T5 is that it is a "text-to-text" model, which means that it is trained to transform the input text into another piece of text. Five distinct variants of T5 have been released: they differ solely for the number of parameters used, ranging from 60 millions to 11 billions. Similarly to other LLMs, T5 models can be fine-tuned for various downstream NLP tasks achieving state-of-the-art results.

In our work, we employ a "t5-base" model (220 million parameters) that has been fine-tuned specifically for conversational question rewriting on the CANARD dataset [12].

2.2.2 HEURISTICS FOR QUERY GENERATION

A different approach consists in mixing the current query text with information coming from previous utterances when generating the query representation. Despite being rather simple, such heuristics are proved to be effective. Across the original submissions of the TREC CAsT 2019 track[8], one fifth employ heuristic rules, providing more than 25% relative nDCG@3 gain w.r.t. submissions not utilizing them.

In this work, we employ the FLC heuristic, consisting of the weighted sum of the textual content of the first (F), last (L) and current (C) utterances. The rationale behind it is that the first utterance often gives the general topic of the conversation, while the previous one is the most likely to be referenced again by the current utterance. This heuristic can synergize with a query rewriting technique, joining its effort to bring contextual information into the current query. FLC is especially useful when the output quality of rewriting is far from ideal.

2.3 TREC CONVERSATIONAL ASSISTANCE TRACK

The Conversational Assistance Track (CAsT) at the Text Retrieval Conference (TREC) was held for the first time in 2019 [8] and since then, at the current

time, it has reached its fourth edition. TREC CAsT has fueled the research in CS domain by providing four large-scale reusable test collections, comprising conversations, corpora, and additional annotations, such as the manual rewrites of the utterances or the canonical response to users' questions [7].

The main task evaluated in CAsT is passages (in 2019 [8] and 2020 [7]) or documents (since 2021 [9]) retrieval from a corpus composed of multiple sub-corpora, such as MS-MARCO [33], TREC CAR [11], Wikipedia [39], and Washington Post corpora.

Since the first edition, the track has evolved towards more natural and human-like conversations, by considerably expanding both the amount and the scope of contextual information that is required by systems to understand a question. For example, in TREC CAsT 2019, user utterances were constructed beforehand by imagining a conversation on a given topic [8], while, from the second edition onward [7, 9] conversations take into consideration the responses given by the system as well. The 2022 edition saw the introduction of a mixed-initiative [1, 22] sub-task, evaluating the ability of systems to produce more engaging and effective conversations, by gaining through feedback questions additional context, details or clarification about the original user utterance.

2.4 IR FRAMEWORKS

Several IR libraries, such as Lucene, Terrier [37], and Anserini [26, 58, 59] allow for extensive and reproducible experimentation with IR systems. These libraries are open-source and were developed for full-text indexing and searching. Furthermore, they can rely on decades of usage, update, and support, as well as flourishing communities underneath. With the advent of ML solutions and the increased popularity of Python, new wrappers around traditional frameworks were designed, such as Pyserini [25] and PyTerrier [30]. Pyserini [25] is a Python toolkit designed as a self-contained package providing a reproducible and easy-to-use first-stage retrieval module within a multi-stage architecture. It supports both sparse and dense representations. PyTerrier [30] is a Python framework developed to allow advanced retrieval pipelines to be expressed and evaluated in a declarative manner. However, none of these frameworks provides native support for conversational search.

We are interested in developing a solution specifically designed for CS but capable of exploiting the advantages of the already existing frameworks. There-

2.4. IR FRAMEWORKS

fore, we opt for a custom implementation based on Java as the main language, while also allowing us to access external Python scripts. In fact, this allows for easy integration with other frameworks, such as Lucene and Anserini, as well as Pyserini and PyTerrier, which are not designed for CS. Furthermore, using Java's strong typing, it is possible to enforce future components implemented within DECAF to follow a rigid external interface. This, in turn, will increase standardization, ease extensibility and reduce the fragmentation in the CS components design.

3

DECAF Architecture

In this section, we provide the overview of DECAF, focusing on, the principal modules of its architecture, components implemented, and software requirements.

3.1 MAIN MODULES

DECAF relies on a modular architecture for index and search pipelines. In practice, to foster extensibility, as well as standardization, each module of DECAF is defined as a Java interface. As illustrated in Figure 3.1, the index pipeline revolves around two main modules: the *Corpus Parser* and the *Indexer*. The former processes the corpus into a stream of documents, which is consumed by the

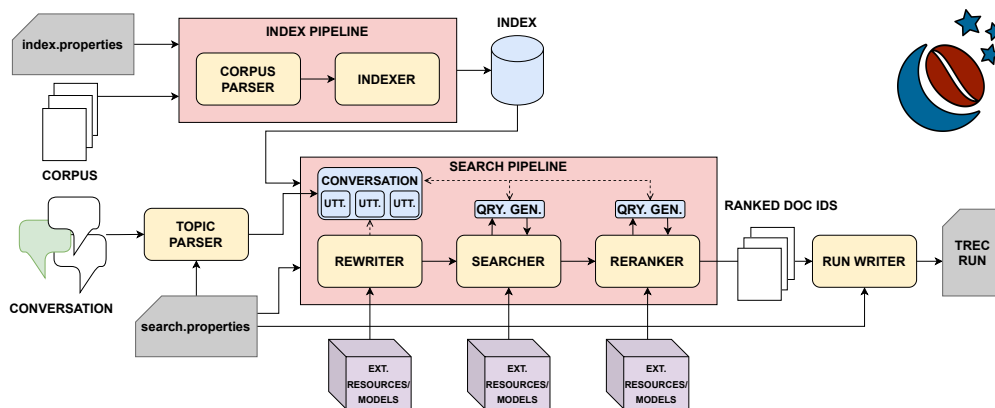


Figure 3.1: Architecture of DECAF.

3.2. COMPONENTS IMPLEMENTED

latter to index them. For the search pipeline, we adopt a multi-stage architecture which employs the modules that are the most common for CS systems.

Figure 3.1 shows the structure of the search pipeline:

- The *Topics Parser* reads in input a file – e.g., written in TREC format – and provides parsed conversations and utterances to be processed by the rest of the framework.
- The *Rewriter* modifies the text of the utterances by adding contextual information extracted from previous utterances in the conversation.
- The *Searcher* takes the (possibly rewritten) utterance text as input, generates the query and retrieves a set of candidate documents to answer the provided question.
- The ranked list of documents generated as output by the Searcher is consumed by the *Reranker*. This module is designed to apply complex and resource-consuming re-ranking operations upon the Searcher output, boosting the performance of the CS pipeline.

Furthermore, we exploit two additional utility models: the Query Generator and the Run Writer. Both the searcher and the re-ranker modules exploit the *Query Generator* to obtain a representation of the user utterance — possibly by combining it with previous ones — that is directly used at retrieval time. Finally, the *Run Writer* is a utility module meant to write the run on a file, so that it can be further used or evaluated.

3.2 COMPONENTS IMPLEMENTED

Every component belonging to a particular module is expected to implement the corresponding Java interface, which defines one or more methods specific to the performed job. The components implemented within DECAF and described in the remainder of this section can be used by practitioners both as baselines as well as templates in extending the framework. Most of them require some configurable parameters, which must be passed through constructor arguments. We also implemented a configuration system, based on *.properties* files, that allows the user to specify them in a user-friendly manner (see Chapter 4).

3.2.1 INDEX PIPELINE

Corpus Parser Three distinct components are implemented to perform corpus parsing. The first processes the passages contained within MS-MARCO version

1¹ dataset. It also supports duplicate removal: the discarded documents are specified through an input file. Another parser addresses the paragraph corpus of TREC CAR v2.0². The third parser processes any corpus based on tab-separated files using the `DOC ID\tDOC TEXT\n'` format.

Furthermore, it might be necessary to index documents from multiple sources with different formats at once. The last parser component eases this by allowing to instantiate multiple parsers that are used to parse different corpora. To provide a practical example, consider TREC CAsT 2019 where both MS-MARCO and TREC CAR corpora were used. This fourth corpus parser provides support in handling this type of scenario, with multiple parsers combined in a single interface.

Indexer The framework comes with three distinct indexer components: the BoW, the SPLADE, and the Dense indexer.

The BoW indexer is a wrapper around Lucene indexing operations. It provides multi-fielded documents, tokenization and advanced analysis capabilities. The efficient inverted index implementation of Lucene makes it suitable for BoW sparse retrieval models. The BoW indexer component has been extended for the SPLADE indexer which is specific to the homonyms neural retrieval model³. It replaces the standard tokenization and analysis pipeline performed by Lucene with the SPLADE model inference. A separate interface to operate with SPLADE is needed since it requires computing first the BoW sparse representations. Custom models based on the same principle, i.e., expanding documents before indexing them, can be instantiated in the same way. Finally, the dense indexer component is specific for dense retrieval models. It is built on top of two well-known libraries: Transformers [52] and Facebook AI Similarity Search (FAISS) [20]. The former is an open-source library providing APIs and tools to download, train and use state-of-the-art ML models. The latter is an open-source library developed for efficient operations, such as clustering, indexing or similarity, on high-dimensional vector spaces. It is particularly efficient and able to handle large datasets composed of billions of vectors with fast query times even in high-dimensional spaces.

¹<https://msmarco.blob.core.windows.net/msmarcoranking/collection.tar.gz>

²<https://trec-car.cs.unh.edu/datareleases/v2.0/paragraphCorpus.v2.0.tar.xz>

³<https://huggingface.co/naver/efficient-splade-V-large-doc>

3.2. COMPONENTS IMPLEMENTED

3.2.2 SEARCH PIPELINE

Conversation and Utterance Components To provide an unified interface to the data, we define two utility components, called *Utterance* and *Conversation*. The former is a data structure that provides unified access to the utterance and subsequent transformations operated by different components of the search pipeline. Instead, the latter provides access and groups together the utterances belonging to the same conversation. At runtime, each module will extract the needed data (e.g., the textual content of the utterance, its rewritten version, or the ranked list associated) from the Utterance component, passing through the Conversation interface. Upon completion of the required operations, each module will save the computed results for a specific utterance (e.g., a new rewriting of the utterance, the re-ranked run), within the Utterance component, so that the next module can access it.

The vast majority of modules for the search pipeline define one or more methods specific to the performed job, which take only two parameters: the Conversation object, containing the data regarding the specific conversation at hand, and the ID of the current utterance on which the component operates (e.g., for rewriting or retrieving the documents for). This approach ensures great flexibility in the component design, since it is possible to access the entire data structure for the whole conversation – in particular to previously issued utterances and retrieved responses. Furthermore, it allows for easily expanding DECAF, since each component can behave as a black-box building block operating only on the Conversation and Utterance objects. The only constraints imposed by the search pipeline are that every module must set some specific data for the current utterance being processed.

Topics Parser DECAF provides five topic parsers designed explicitly to handle TREC CAsT 2019 and 2020 evaluation topics. More in detail, for each collection, DECAF has a parser for each type of utterance – i.e., either manual or automatic utterances. We design two parsers specifically for the automatic and manual utterances for TREC CAsT 2019. Moreover, we also implemented the equivalent ones for TREC CAsT 2020. The fifth parser, instead, gives the automatically rewritten utterances for the second edition, which were produced using an automatic method by the organizers. Notice that, we define such a high number of different topic parsers components since they also contain specific preprocess-

ing operations – this allows us to feed directly the original topic files to DECAF.

This component takes in input the topics file – using, for simplicity and to ease reproducibility, directly those provided by TREC CAsT organizers – and generates in output a stream of Conversation objects. Each of them is further split into the individual Utterance objects that compose it.

Rewriter DECAF provides two state-of-the-art rewriting approaches using off-the-shelf resources, either employing coreference resolution libraries or pre-trained T5 models.

In implementation terms, there are two distinct components that carry out coreference resolution, which differ solely for the library used to perform the operation. In particular, one component is based on AllenNLP framework [17], while the other uses Fastcoref. Fastcoref is a coreference resolution utility based on the LingMess [36] architecture, providing state-of-the-art coreference accuracy [35].

For the second approach, a T5 model is employed, which is a large-scale, unsupervised text-to-text transfer learning model that relies on the transformer architecture and can be fine-tuned for various NLP tasks, including anaphora and coreference resolution [43]. The particular instance of T5 used in the experimental part is publicly available and pre-trained specifically for conversational search question rewriting⁴.

To maintain the modularity of the framework, we also implement a rewriter which corresponds to the “identity” operation and returns the original text unchanged. It should be used in all cases where the utterances have been rewritten externally from the framework or if the practitioner does not wish to carry out any form of rewriting.

The rewriter expands the original text of the utterance into the rewritten text and stores it within the specific Utterance object.

Query Generator The whole conversation is considered as input for components implemented within this module, producing as output a representation of the utterance that embeds the context. Within DECAF, we provide three different query generator components.

⁴<https://huggingface.co/castorini/t5-base-canard>

3.2. COMPONENTS IMPLEMENTED

The FLC query generator takes in input the utterances for a given conversation and outputs a weighted sum of the rewritten text of the first (F), last (L) and current (C) utterances. The rationale behind it is that the first utterance often gives the general topic of the conversation, while the previous one is the most likely to be referenced again by the current utterance. The output of this component is meant to synergize with the rewriter’s effort to bring contextual information into the current query, especially useful when the quality of its output is less than ideal. The overall effectiveness of such a simple approach in modeling the context has been already observed [32].

The Sequence query generator provided within DECAF considers the concatenation of the rewritten text for all previous utterances of the current conversation. Instead, the Current query generator considers only the – possibly rewritten – text of the current utterance, without taking into account any of the previous ones.

To operate, the query generator accesses the Utterance objects referring to the current and – possibly – the previous utterances.

Searcher The searchers are specular to the indexers used in the indexing phase, with three different searcher components, one for BoW Lucene-based similarity functions, one for SPLADE and one for dense models.

The first searcher component, the BoW one, by being based on Lucene, can be instantiated with any of the classical BoW models already implemented in it, such as BM25 [45], LM [65] or the Vector Space Model [46]. For example, in the experimental benchmarking reported in Chapter 5, we exploit BM25 [45]. It is a BoW IR model that ranks documents based on the occurrence and frequency of terms. Notice that the lexical similarity function used can be set at runtime (see Chapter 4).

The second similarity function taken into consideration is SPLADE [15, 16]. It is a NIR model that learns sparse representation for both queries and documents via the BERT MLM head and sparse regularization. SPLADE is particularly appealing for the first stage retrieval phase, thanks to the simplicity, efficiency, and explainability of sparse representations. We separated it from the previous component, even though SPLADE is a BoW model, because SPLADE requires computing the BoW sparse representation of the query. In our experimental

analysis, we utilize a publicly available set of weights⁵.

Finally, we implement a component for dense retrieval that can be used with FAISS indexes. In particular, it is instantiated with BERT, which is a pre-trained LLM and it has been trained to learn dense representations of words from unlabeled text, by jointly conditioning on both left and right context [10]. Notice that, for the experimental analysis, we exploit a publicly available BERT instance fine-tuned specifically for IR⁶.

Components implementing this module invoke the query generator sub-component that provides them with a searcher-specific query representation directly used for retrieval. Upon retrieval completion, components within the searcher module must save the top- k retrieved document IDs within the Utterance component.

Reranker Components implemented within the reranker module consider the documents included in the ranking list produced by the searcher and generate a new relevance score for each of them.

At the current time, DECAF comes with a reranker component: Transformers. The Transformers reranker employs the Transformers (Hugging Face) [52] library to apply ML models, such as BERT, to the text of both the rewritten query and of the documents retrieved by the searcher, then evaluate the similarity between them. At the run-time, the Transformers component can be customized depending on the practitioner’s needs through the *properties* file. About the transformers model employed, the experiments focus on BERT, since several works already observed its effectiveness for the re-ranking task in the CS and IR domains [32, 34, 29]. It is possible to set different similarity functions, enabling to adapt to different models. The similarity functions currently implemented are cosine similarity, dot product and Euclidean distance. This component can optionally perform run fusion between the newly-generated ranked list with the one given by the searcher.

Finally, we assume that a user might not be interested in re-ranking documents. In that case, we included the identity reranker which returns the ranked list of documents generated by the searcher.

⁵<https://huggingface.co/naver/efficient-splade-V-large-query>

⁶https://huggingface.co/sebastian-hofstaetter/distilbert-dot-tas_b-b256-msmarco

3.3. EXTERNAL PROCESS INTEGRATION

This module must set the final ranked list of documents in the Utterance object. This final ranking is the output of the whole search pipeline for the current utterance.

Run Writer This final module can be instantiated into two modalities. The first component – dubbed Trec Eval – produces a run using the standard TREC eval format. In particular, it saves inside the *runs* sub-folder of the framework a tab-separated file with six columns: the query id, the Q0 placeholder, the document id and ranking, the retrieval score, and the user-specified id of the run. Secondly, to ease the debugging, the “debug” component saves on file both ranked lists produced by the searcher and reranker together with a file containing the top-*k* documents retrieved by the system to allow for manual inspection and precise failure analysis.

3.3 EXTERNAL PROCESS INTEGRATION

Some components implemented out-of-the-box integrate external Python code within the main architecture developed in Java. The objective of this section is to provide a detailed explanation on how this integration is achieved and what operations are performed, in order to supply practitioners with the knowledge required to extend the framework. Despite our codebase exemplifies this method using only Python code, the same methodology can be employed to integrate any external process.

Figure 3.2 shows the three main steps and the flow of data exchanged by the Java and external processes.

3.3.1 CREATION

In the initial phase, the main Java process creates the external one as a daemon process, which operates as a background process, running continuously without any direct interaction with the user until the main process is terminated. The two processes can communicate with each other and exchange data through their input, output and error streams. Further information is provided in the following subsection.

Within DECAF, the *ExternalScriptDriver* Java utility class is employed to manage the entire lifecycle of the external process, but also to handle its streams. The

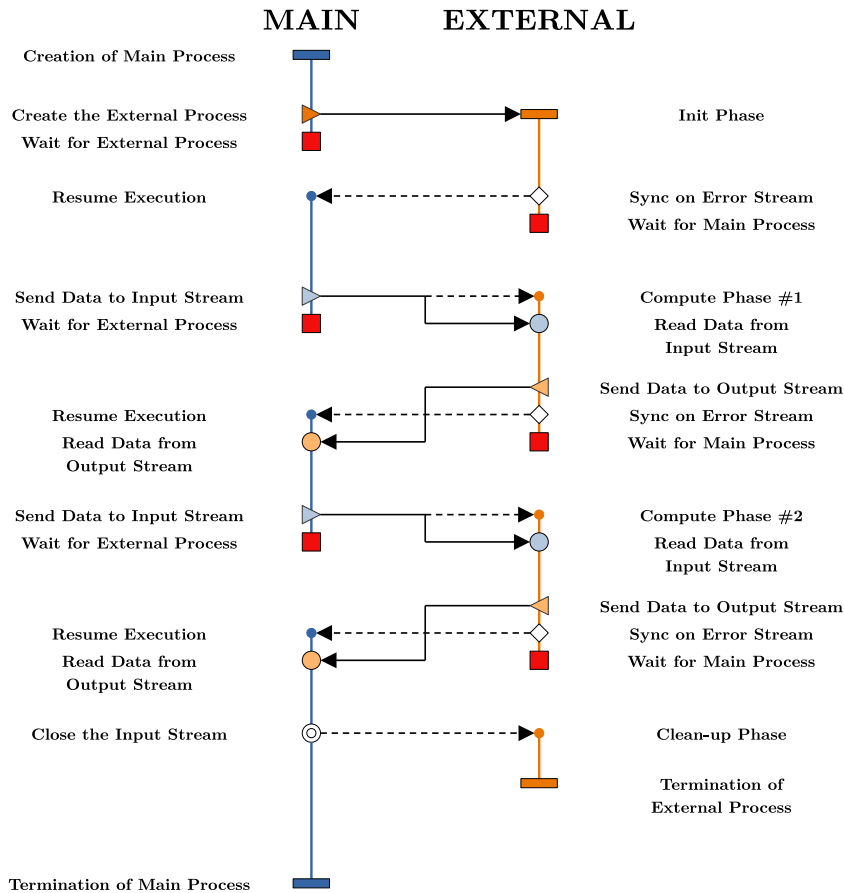


Figure 3.2: Typical data and execution flow between the Main and External processes in DECAF.

external process is created inside the constructor. Additionally, it is possible to specify its working directory. When the constructor has finished, the two processes are both alive and being executed concurrently.

3.3.2 SYNCHRONIZATION AND DATA EXCHANGE

The main and the external processes are scheduled independently from each other. Except for basic tasks, the two processes are required to synchronize and exchange data to successfully deliver the expected result(s). To achieve this, the streams of the external process are employed.

The main process send/receive data to/from the external process through its input and output stream, respectively. Instead, the error stream is employed for synchronization purposes. The main Java process awaits for text on the error stream: in case of an empty line, no errors have occurred, hence the external

3.3. EXTERNAL PROCESS INTEGRATION

process has reached the expected state, otherwise an exception has occurred, therefore the main process is terminated after displaying the error message to the user. Immediately after this step has taken place, it is the appropriate time for the main process to retrieve data from the output stream. It is imperative to synchronize on the error stream, since waiting on the output stream may result in the main process hanging indefinitely in the event of an exception being thrown in the external process. The external process, on the other hand, awaits new data to be processed from its input stream. No additional synchronization is required in this case: both the external and main processes are terminated when an unhandled exception is thrown in the main process. Note that, to ensure that the receiver process obtains the data sent through a stream, it is necessary to flush it after the transmission.

Operationally, the *ExternalScriptDriver* provides several convenient methods. The `hasNextOutput/ErrorLine` method verify whether the external process has written any data to its output or error stream. The content can be retrieved using the `nextOutput/ErrorLine` and `nextOutput/ErrorText` methods, which differ in the type of data returned: the former provided the subsequent line of text, while the latter provides the entire block of text available. Finally, the `waitNextOutput/ErrorLine` and `waitNextOutput/ErrorText` methods can also suspend the current thread until the data becomes available.

3.3.3 CLEAN-UP

The external process will continue to expect data from the input stream until it is closed by the main process. The `getInputStream` method of the *ExternalScriptDriver* class is employed to retrieve the input stream object, then its `close` method is called to shut down the stream. As a consequence, the external process should perform any necessary clean-up tasks, such as releasing any resource in use, and then terminate. Typically, at this point no further interaction is required between the processes.

3.3.4 TECHNIQUES TO AVOID STREAM CLUTTERING IN PYTHON

One crucial aspect of the synchronization mechanism outlined in the preceding subsections is that the external process employs the error stream to convey its status to the main process. Unfortunately, numerous developers tend to clutter the error stream with debug information, progress status, and other insignifi-

cant log data. This, in turn, make it challenging to integrate many pre-existing libraries into our framework.

In this subsection, we describe the principal techniques used in Python to suppress such superfluous information. The objective is to help practitioners avoiding the same difficulties we faced while implementing the components available out-of-the-box in DECAF that they might encounter while developing their own components. Note that, in case of the "logging" and "tqdm" libraries, the code reported here must be placed before the import statement of problematic libraries that contribute to the cluttering of output and/or error streams.

Disable the "logging" Library

```
import logging
logging.disable(logging.CRITICAL)
```

Disable the "tqdm" Library

```
import functools
import tqdm
tqdm.tqdm.__init__ = functools.partialmethod(tqdm.tqdm.__init__,
    disable = True)
```

Disable the "warnings" Library

```
import warnings

# Suppress the warnings using the following command:
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    # Problematic code goes here.
```

Suppress output to a Stream

```
import contextlib
import os

@contextlib.contextmanager
```

3.4. GIT REPOSITORY

```
def suppress_stream(stream):
    try:
        old_fd = os.dup(stream.fileno())
        fnull = open(os.devnull, 'w')
        os.dup2(fnull.fileno(), stream.fileno())
        yield
    finally:
        if old_fd is not None:
            os.dup2(old_fd, stream.fileno())
        if fnull is not None:
            fnull.close()

# Suppress the error stream using the following command:
with suppress_stream(sys.__stderr__):
    # Problematic code goes here.
```

3.4 GIT REPOSITORY

DECAF is available as open source under the *Creative Commons Attribution-ShareAlike 4.0 International License*⁷ at the following address: <https://github.com/alemarco96/DECAF>.

As shown in Figure 3.3, DECAF comes with extensive documentation and guides to help new users adopt the framework. There are tutorials dedicated to the installation process and the setup of configuration files for conducting experiments. Regarding the reproducibility of the experiments shown in Chapter 5, we included a detailed guide but also, for each of them, both the configuration file employed as well as the run generated. This data is located inside the *SIGIR-experiments* directory.

The framework is organized around several directories, each with a different purpose, as shown in Figure 3.4. We describe here such directories – to ease the framework instantiation, they can be found on the *template* sub-directory of DECAF repository⁸.

⁷<http://creativecommons.org/licenses/by-sa/4.0/>

⁸In case of highly customized scenarios, the paths to these directories, as well as specific files, could also be manually set through the configuration files and environmental variables declared

alemarco96 / DECAF Public

Repository of the DECAF framework, described in the "DECAF: a Modular and Extensible Conversational Search Framework" paper.

☆ 0 stars 🍴 0 forks

☆ Star Unwatch

<> Code Issues Pull requests Actions Projects Wiki Security ...


main

alemarco96 Version 1.0.0 upload. 1 minute ago 14

📁 SIGIR2023-experiments	yesterday
📁 logos	yesterday
📁 setup_scripts	1 minute ago
📁 src/main	yesterday
📁 template	yesterday
📄 .gitignore	last week
📄 CONFIGURATION_GUIDE.md	last week
📄 INSTALLATION_GUIDE.md	1 minute ago
📄 README.md	1 minute ago
📄 REPRODUCIBILITY_GUIDE.md	1 minute ago
📄 allennlp_spacy_transformers.yml	last week
📄 faiss_fastcoref_spacy_transformers.yml	last week
📄 pom.xml	last week

README.md

DECAF



DECAF is a framework for performing conversational search. Its modular architecture enables it to easily extend and adapt to suit most applications' needs while reusing most of the pre-defined components. The core of DECAF is written in Java, while some components are implemented in Python. All machine-learning code supports acceleration through CUDA-enabled GPU. It is developed at the [Intelligent Interactive Information Access Hub, Department of Information Engineering \(DEI\), University of Padua](#).

Figure 3.3: Screenshot of the main directory of DECAF repository.

3.4. GIT REPOSITORY

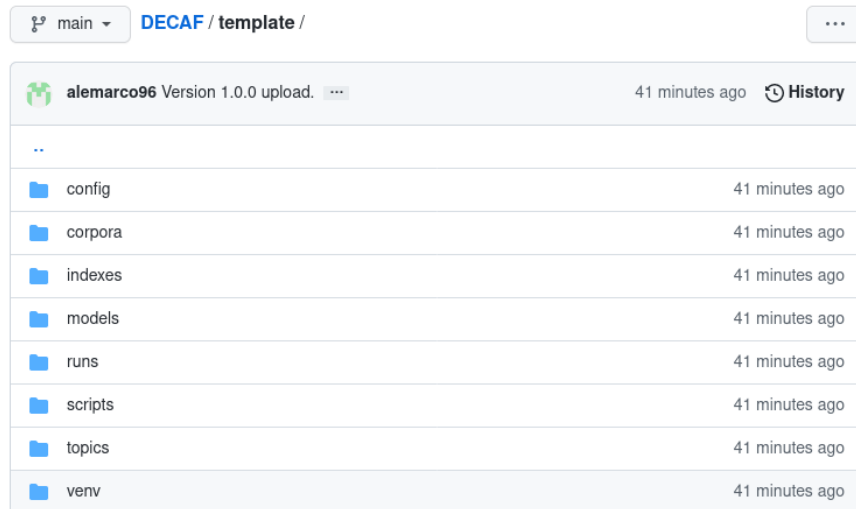


Figure 3.4: Screenshot of the *template* sub-directory of DECAF repository.

- The *config* folder contains the properties files defining which components are instantiated along with their parameters.
- The *corpora* folder contains the corpora used for indexing.
- The *indexes* folder contains the indexes created by DECAF, upon completion of the index pipeline.
- The *models* folder should be filled by the practitioner with the machine-learning models employed.
- The *runs* folder contains the runs produced DECAF upon completion of the search pipeline.
- The *scripts* folder contains the scripts used to execute either the index or the search pipeline.
- The *topics* folder contains the evaluation topics files.
- The *venv* folder contains the Python virtual environments employed by the components.

To operationalize DECAF, it is first necessary to install it by compiling the Java code using the Maven project management tool. Then, if needed, it is necessary to install all the required Python modules within the *venv* directory and download the models chosen by the user in the *models* directory. Finally, it is necessary to download and place all corpora and topics within the *corpora* and

in the launch scripts.

topics directories respectively. Notice that, we do not provide any collection, since most of them require practitioners to accept “Terms and Conditions”. After that, it is possible to run DECAF, either by using the *properties* files already available within the *config* directory or define new configuration files. It is possible to use DECAF by running `scripts/index.sh` and `scripts/search.sh` to run indexing and searching respectively. Output runs will be placed in the *runs* directory.

4

DECAF in Action

4.1 INSTALLATION GUIDE

4.1.1 REQUIREMENTS

The core of DECAF has been developed in Java, with the integration of Python for ML-oriented functionalities. It requires Java Development Kit (JDK) 11 and Python 3.8 for execution. The framework employ Maven together with Conda and Pip to manage the external dependencies in Java and Python, respectively. DECAF is built upon Lucene 8.8.1¹.

Additionally, at least one Nvidia GPU with support for CUDA 11.3 is highly recommended to significantly speed-up the execution of many components relying of ML functionalities.

4.1.2 DEPENDENCIES

DECAF and its dependencies can be installed using two distinct methods, either directly on the host machine or within a Singularity container. The latter approach involves creating a container image that includes all the necessary software require to operationalize the framework, which can subsequently be deployed on various hardware platforms. This solution is particularly suitable for executing DECAF within a computing cluster, such as the one available at

¹https://lucene.apache.org/core/8_8_1/index.html

4.1. INSTALLATION GUIDE

DEI. For the remainder of this guide, we assume that the operating system used is a Debian-based distribution of Linux, such as Ubuntu.

INSTALL SYSTEM DEPENDENCIES

The following libraries must be installed first:

```
sudo apt install -y git git-lfs wget
```

SINGULARITY

Singularity is a free and open-source containerization software designed specifically for high-performance computing applications, available exclusively for Linux operating system. It provide native support for high-performance interconnections and PCIe-attached devices, such as graphics accelerators. These features are particularly valuable for any computation-intensive workloads, such as ML and, in general, for any data-driven application. Furthermore, Singularity enables reproducibility to scientific computing, by providing complete environments that can easily be copied and executed on many different hardware platforms.

Install System Dependencies The following development tools and libraries must be installed first:

```
sudo apt install -y build-essential libseccomp-dev libglib2.0-dev \
pkg-config squashfs-tools cryptsetup crun uidmap
```

Install Go Compiler Singularity is a software implemented in Go that requires to be installed from source. Therefore, the Go compiler is a necessary prerequisite for installation. In our work, we employed the 1.19.2 version.

```
# Change these variables to reflect your requirements.
```

```
GO_VERSION=1.19.2
```

```
CPU_ARCH=amd64
```

```
FILENAME=go${GO_VERSION}.linux-${CPU_ARCH}.tar.gz
```

```
wget -O /tmp/${FILENAME} https://dl.google.com/go/${FILENAME}
```

```
sudo tar -C /usr/local -xzf /tmp/${FILENAME}
```

Make sure to add `/usr/local/go/bin` to the `PATH` environmental variable:

```
echo 'export PATH=$PATH:/usr/local/go/bin' >> ~/.bashrc
source ~/.bashrc
```

Install Singularity from Source Clone the repository of Singularity in a location of your choice, then proceed to build and install using the following commands. In our work, we employed the 3.10.3 release.

```
# Change this variable to reflect your requirements.
SINGULARITY_VERSION=3.10.3

# Clone the repository and select the desired version.
git clone --recurse-submodules \
https://github.com/sylabs/singularity.git
cd singularity
git checkout --recurse-submodules v${SINGULARITY_VERSION}

# Build and install Singularity.
./mconfig
make -C builddir
sudo make -C builddir install
```

Generate the Singularity Container The Singularity container employed in our work is based on an image of Ubuntu 22.04. Open the terminal, navigate to a location of your choice, and insert the following command:

```
CONTAINER_NAME=decaf
CONTAINER_LOCATION=/path/to/desired/location

cd $CONTAINER_LOCATION
singularity build --sandbox ${CONTAINER_NAME} docker://ubuntu:22.04
```

After the container has been successfully created, all the dependencies must be installed inside it. Therefore, every command explained in the remainder of this section should be typed inside a shell operating on the container. Note that some commands require super-user privileges on the container, while others not. In the former case, `sudo` must be prepended before the shell command.

4.1. INSTALLATION GUIDE

```
# Shell operating on the container, super-user privileges.
sudo singularity shell --writable ${CONTAINER_NAME}/

# Shell operating on the container, default privileges.
singularity shell --writable ${CONTAINER_NAME}/

# Type the commands on the obtained shell.

# Command to exit the shell.
exit
```

After all dependencies (Java, Maven, Python, Conda, and all Python virtual environments) have been installed, the container is finalized into the read-only .sif image using the following command:

```
sudo singularity build ${CONTAINER_NAME}.sif ${CONTAINER_NAME}/
```

The result of this procedure consists in the definitive container image, which can be employed in different machines to replicate the same software setup. Sometimes, the need to deploy our setup in an environment where we lack super-user privileges may arise. In such cases, it is possible to build the container on another machine, and then copy the definitive image on the target platform where Singularity have already been installed by system administrators.

JAVA DEVELOPMENT KIT AND MAVEN

The core of DECAF has been developed in Java, and require Maven to compile and generate the JAR file. In our work, we employed the JDK 11.

```
sudo apt install openjdk-11-jdk
sudo apt install maven
```

PYTHON AND CONDA

Some components of DECAF are partially implemented on Python, using external packages managed by Conda and Pip. In our work, we employed Mini-conda 4.12.0 with Python 3.8 which includes all required software.

```
# Change these variables to reflect your requirements.
CONDA_VERSION=4.12.0
CPU_ARCH=x86_64

FILENAME=Miniconda3-py38_${CONDA_VERSION}-Linux-${CPU_ARCH}.sh
wget https://repo.anaconda.com/miniconda/${FILENAME}
chmod +x ${FILENAME}
./${FILENAME}
```

4.1.3 INSTALLATION

BUILD DECAF FROM SOURCE

Clone the repository of DECAF in a location of your choice, then proceed to build the main executable JAR file using Maven:

```
cd /path/to/desired/location
git clone https://github.com/alemarco96/DECAF.git
cd DECAF
DECAF_ROOT_FOLDER="$(pwd)"

mvn package
```

Optionally, generate the JavaDoc documentation of the source code, which will be placed inside the *javadoc* subdirectory of the root folder:

```
JAR_FILENAME=$(ls target/ | grep ".*-jar.*.jar")
javadoc -protected -d javadoc -sourcepath src/main/java \
  -subpackages it.unipd.dei -classpath target/${JAR_FILENAME}
```

Notice that, if a Singularity container is employed, this folder must not be placed inside the container but, instead, in a user-decided location within the filesystem of the machine where DECAF is deployed. Otherwise, it won't be possible to export any data produced by DECAF, as the container filesystem is read-only.

4.1. INSTALLATION GUIDE

SETUP PYTHON VIRTUAL ENVIRONMENTS

The components available out-of-the-box in DECAF assume that three Python virtual environments have been created as detailed in this subsection.

If a Singularity container is employed, their data must be stored in a user-decided location within the container filesystem (for example, use `/venv`). Otherwise, all data regarding them should be stored inside the `template/venv` subdirectory.

```
# Update conda and pip to a recently-released version.
conda update -n base -c defaults conda
pip install pip --upgrade pip

# If Singularity container is employed.
DECAF_VENV_FOLDER=/venv
# If DECAF is installed locally.
DECAF_VENV_FOLDER="$DECAF_ROOT_FOLDER/template/venv"

# Create the "allennlp_spacy_transformers" environment:
VENV_NAME=allennlp_spacy_transformers

conda env create -f $VENV_NAME.yml -p $DECAF_VENV_FOLDER/$VENV_NAME
conda activate $DECAF_VENV_FOLDER/$VENV_NAME
python -m spacy download en_core_web_sm
conda deactivate

# Create the "faiss_fastcoref_spacy_transformers" environment:
VENV_NAME=faiss_fastcoref_spacy_transformers

conda env create -f $VENV_NAME.yml -p $DECAF_VENV_FOLDER/$VENV_NAME
conda activate $DECAF_VENV_FOLDER/$VENV_NAME
python -m spacy download en_core_web_sm
conda deactivate

# Create the "splade" environment:
wget https://raw.githubusercontent.com/naver/splade/main/\
  conda_splade_env.yml
```

```
conda env create -f conda_splade_env.yml \
  -p $DECAF_VENV_FOLDER/splade
```

DOWNLOAD ALL MODELS

DECAF requires to manually download the ML models used by the default components. All data regarding them are stored inside the *template/models* sub-directory.

```
DECAF_MODELS_FOLDER="$DECAF_ROOT_FOLDER/template/models"

# Keep separated AllenNLP and Transformer-based models.
mkdir $DECAF_MODELS_FOLDER/allennlp
mkdir $DECAF_MODELS_FOLDER/transformers

# AllenNLP-based "coref-spanbert-large" model:
MODEL_ARCHIVE=coref-spanbert-large-2021.03.10.tar.gz
MODEL_LOCATION=$DECAF_MODELS_FOLDER/allennlp/coref-spanbert-large

mkdir $MODEL_LOCATION
wget -p /tmp https://storage.googleapis.com/allennlp-public-models/\
  $MODEL_ARCHIVE
tar -xf $MODEL_ARCHIVE -C $MODEL_LOCATION
rm /tmp/$MODEL_ARCHIVE

# Transformers-based models.
cd $DECAF_MODELS_FOLDER/transformers

git lfs clone https://huggingface.co/\
  distilbert-dot-tas_b-b256-msmarco
git lfs clone https://huggingface.co/efficient-splade-V-large-doc
git lfs clone https://huggingface.co/efficient-splade-V-large-query
git lfs clone https://huggingface.co/f-coref
git lfs clone https://huggingface.co/multi-qa-mpnet-base-dot-v1
git lfs clone https://huggingface.co/t5-base-canard
```

4.2. CONFIGURATION GUIDE

EDIT THE LAUNCH SCRIPTS

The launch scripts provided with DECAF expect to find the resources in locations defined by specific environmental variables. Practitioners are required to manually edit both `index.sh` and `search.sh` launch scripts, which are located inside the `template/scripts` subfolder, to make sure these environmental variables reference the correct locations on disk. Notice that every location must be referenced with their absolute path. If practitioners employ the default locations suggested during the installation guide, only a few edits are required. In line 7, the `DECAF_ROOT_FOLDER` must correspond to the location where DECAF has been cloned. In line 19, `DECAF_VENV_FOLDER` must be set to the same value used before during the setup of Python virtual environments. If practitioners employ a Singularity container, in line 15, `DECAF_CONTAINER_ROOT_FOLDER` must be set with a user-decided absolute location where the root folder of DECAF is made available within the container filesystem. Moreover, in line 41 the following directive must be added before the command to execute:

```
singularity exec --nv $CONTAINER_LOCATION/$CONTAINER_NAME.sif \  
  --bind $DECAF_HOST_ROOT_FOLDER:$DECAF_CONTAINER_ROOT_FOLDER \  
  <command to execute>
```

where “`$CONTAINER_LOCATION`” and “`$CONTAINER_NAME`” must be replaced with the same values used during the creation of Singularity container image, as was detailed in the installation procedure.

4.1.4 UNINSTALL

The procedure for uninstalling the framework slightly differs, based of whether a Singularity container has been employed or not. In the former case, delete the definitive `.sif` container image which was built following the instructions detailed in the previous section. In both cases, delete the root folder of DECAF. This will remove all source code, documentation, data, and models.

4.2 CONFIGURATION GUIDE

We describe here the procedure required to configure DECAF in order to execute them.


```

1 launch.corpus = CAsT1920
2
3 # Path to the corpus files
4 launch.corpus.CAsT1920.msmarco_corpus_filename =
5 /path/to/location
6 launch.corpus.CAsT1920.msmarco_duplicate_filename =
7 /path/to/location
8 launch.corpus.CAsT1920.treccar_corpus_filename =
9 /path/to/location
10
11 launch.indexer = BoWIndexer
12
13 # Path to the directory containing the index
14 launch.indexer.BoWIndexer.index_directory =
15 /path/to/location
16
17 # Text normalization component
18 launch.indexer.BoWIndexer.analyzer = English
19
20 # Similarity function used and other parameters
21 launch.indexer.BoWIndexer.similarity = BM25
22 launch.indexer.BoWIndexer.similarity.BM25.k1 = 0.82
23 launch.indexer.BoWIndexer.similarity.BM25.b = 0.4
24 launch.indexer.BoWIndexer.chunks_size = 1000000
25
26 launch.num_threads = 8
27

```

Configuration 4.1: index.properties

Figure 4.1: An example of configuration file snippet that allows for configuring the indexing phase.

The settings are composed of *properties* files, one specific for index and another for search phases, responsible to specify which components are instantiated together with their parameters. The proper use of these files allows the execution of each pipeline according to the desired experimental setting.

The *properties* is a human-readable language commonly used in Java projects for configuration files. Each line represents a key=value pair.

Notice that, the *properties* file is divided into two parts: the upper part contains more volatile information (models used, paths, parameters), while the lower part contains boilerplate and advanced settings that, in a ready-to-use scenario, can remain unchanged for the basic user.

4.2.1 INDEX PHASE

4.2. CONFIGURATION GUIDE

The first operation is to index all the documents, which can be customized using the provided `index.properties` file located inside the `config` sub-folder. Figure 4.1 presents a minimal working example of how configuring the `index.properties` file.

The `launch.corpus` parameter is responsible for selecting which documents corpus will be indexed. The option `CAsT1920` allows for replicating the results reported in Chapter 5. Notice that, the specific component identified by the name `CAsT1920` identifies a specific multiple corpora parser that combines the `MS-MARCO` and `TREC CAR` parsers. With `launch.indexer` it is possible to specify the indexer that will perform the indexing operation. There are some additional parameters to set for this component. The `index_directory` is the absolute path to the location on the disk where all index data is stored. Moreover, the documents are processed in chunks of size given by `chunks_size` parameter. Other parameters, such as `BoW.analyzer`, are specific for the specified indexer. Lastly, `launch.num_threads` determines the number of CPU cores used to speed-up the execution of this phase.

4.2.2 SEARCH PHASE

In this section, we describe how to customize the search phase operations using the `search.properties` configuration file. Figure 4.2 shows a minimal example of it, performing first stage retrieval using `BM25` Bag-of-Words model then re-ranking with a `BERT` model. The `launch.topics` allows to choose which evaluation topics are processed. The desired rewriter component can be picked with the `launch.rewriter` parameter. Furthermore, the `launch.searcher` selects which searcher to use. Note that each one require that the documents have been already been processed by the corresponding indexer before attempting execution. The `index_directory` sub-parameter specifies the absolute path to the folder where the index data have been stored. Another important parameter is `query`: it allows to customize the query representation used to perform retrieval. The desired re-ranker can be selected through the `launch.reranker` property. The `launch.run_writer` component selects how to consume the results produced up to that point; `TrecEval` options generate a standard run using `TREC-Eval` format. The `run_id` sub-parameter defines the identifier of the run. In conclusion, `launch.num_documents` and `launch.num_threads` let you pick the maximum number of documents included in the results for each query and the

number of CPU cores used to speed-up execution, respectively.

4.2. CONFIGURATION GUIDE

```
1 launch.topics = AutCAsT19
2
3 launch.rewriter = T5
4 launch.rewriter.T5.model = t5-base-canard
5 launch.rewriter.T5.max_tokens = 512
6
7 launch.searcher = BoWSearcher
8
9 # Path to the directory containing the index
10 launch.searcher.BoWSearcher.index_directory =
11 /path/to/location
12
13 # Text normalization component
14 launch.searcher.BoWSearcher.analyzer = English
15
16 # Similarity function used and its parameters
17 launch.searcher.BoWSearcher.similarity = BM25
18 launch.searcher.BoWSearcher.similarity.BM25.k1 = 0.82
19 launch.searcher.BoWSearcher.similarity.BM25.b = 0.4
20
21 # The query generator component
22 launch.searcher.BoWSearcher.query = Current
23
24 launch.reranker = Transformers
25
26 # The model to use with its parameters
27 launch.reranker.Transformers.model =
28 distilbert-dot-tas_b-b256-msmarco
29 launch.reranker.Transformers.vector_size = 768
30 launch.reranker.Transformers.max_tokens = 512
31 launch.reranker.Transformers.similarity = dot
32
33 # The query generator component
34 launch.reranker.Transformers.query = Current
35
36 # The fusion component
37 launch.reranker.Transformers.fusion = No
38
39 launch.run_writer = TrecEval
40 launch.run_writer.TrecEval.run_id = id_of_the_run
41
42 launch.num_documents = 100
43 launch.num_threads = 8
44
```

Configuration 4.2: search.properties

Figure 4.2: An example of a configuration file snippet that allows for configuring the search phase.

5

Experimental Results

The framework has been tested on TREC CAsT 2019 and 2020 settings, two popular benchmarks for evaluating Conversational Information Seeking (CIS) systems. We experiment with multiple combinations of components and parameters, summarized in Tables 5.1, 5.2, and 5.3. Following the procedure used for CAsT evaluation campaign [8, 7], we report four widely used measures: R@100, MRR, nDCG@3, and nDCG@10, computed using the *trec_eval*¹ tool. Following the official evaluation settings on TREC CAsT 2020 dataset, we consider documents as relevant if their relevance score is ≥ 2 [7].

Table 5.1: Rewriters and corresponding configuration parameters, used in the evaluation of DECAF.

ID	Rewriter conf.	Description
—	rewriter: No	No rewriting is applied at all.
A-CR	rewriter: AllenNLP model: coref-spanbert-large	Apply co-reference resolution, replacing most expressions that refer to the same entity.
F-CR	rewriter: Fastcoref model: f-coref	Apply co-reference resolution, replacing most expressions that refer to the same entity.
T5	rewriter: T5 model: t5-base-canard	T5 model trained specifically for conversational search question rewriting.

¹https://github.com/usnistgov/trec_eval

Table 5.2: Searchers (first-stage ranking functions) and corresponding configuration parameters, used in the evaluation of DECAF.

ID	Searcher conf.	Description
BM25 _c	Indexer: BoW Analyzer: English Similarity: BM25 Query generator: Current	First stage retrieval using BM25 Bag-of-Words model. The query representation is built considering only the rewritten text of the Current utterance.
BM25 _{FLC}	Indexer: BoW Analyzer: English Similarity: BM25 query generator: FLC	First stage retrieval using BM25 Bag-of-Words model. The query representation is built considering the rewritten text of the First, Last and Current utterances.
DIR. _c	Indexer: BoW Analyzer: English Similarity: Dirichlet Query generator: Current	First stage retrieval using Dirichlet LM Bag-of-Words model. The query representation is built considering only the rewritten text of the Current utterance.
DIR. _{FLC}	Indexer: BoW Analyzer: English Similarity: Dirichlet query generator: FLC	First stage retrieval using Dirichlet LM Bag-of-Words model. The query representation is built considering the rewritten text of the First, Last and Current utterances.
BERT _c	Indexer: Dense Model: distilbert-dot-tas_b-b256-msmarco (BERT) Similarity: dot product Query generator: Current	First stage retrieval using BERT model for dense retrieval. The query representation is built considering only the rewritten text of the Current utterance.
SPLADE _c	Indexer: Splade Model: efficient-splade-V-large-query Query generator: Current	First stage retrieval using SPLADE Bag-of-Words model. The query representation is built from the rewritten text of the Current utterance.

Table 5.3: Rerankers and corresponding configuration parameters, used in the evaluation of DECAF.

ID	Reranker conf.	Description
—	reranker: No	No re-ranking is applied at all.
BERT	reranker: Transformers model: distilbert-dot-tas_ b-b256-msmarco (BERT) similarity: dot query generator: Current fusion: No	Re-ranking is performed using a BERT model. Each document is compared against the query, by applying dot-product similarity function between the embeddings produced by the model.

Table 5.4: List of experiments conducted on TREC CAsT 2019 benchmark, testing different rewriting strategies and query generation heuristics, detailed in Section 5.1.1. The evaluation measures reported are R@100, MRR, and nDCG with cutoffs 3 and 10.

	Topics	Rew.	Query Gen.	Searcher	Rec.	MRR	nDCG@3	nDCG@10
1	Auto.	—	C	BM25	19.9	32.0	14.2	14.4
2		A-CR	C		34.0	49.9	24.7	24.2
3		F-CR	C		35.3	51.5	25.9	25.4
4		T5	C		42.8	64.0	33.9	32.1
5		—	FLC		30.6	48.3	23.6	23.0
6		A-CR	FLC		41.8	59.1	29.7	29.3
7		F-CR	FLC		41.6	61.2	31.0	30.1
8		T5	FLC		44.2	67.4	35.6	33.0
Baseline Automatic at TREC CAsT 2019					21.4	31.8	14.8	15.9
9	Man.	—	C	BM25	47.8	66.7	35.4	34.5
10		FLC	49.5		70.2	36.6	35.2	

5.1 TREC CAsT 2019 RESULTS

In this section, we show the experimental results obtained on TREC CAsT 2019 evaluation campaign. Firstly, we test the performance impact of multiple components implemented within DECAF, then we focus on those pipelines that best performed throughout the initial experiments.

5.1.1 REWRITERS AND QUERY GENERATION HEURISTICS

Table 5.4 shows the experimental results on TREC CAsT 2019, obtained testing different rewriting strategies and query generation heuristics, employing the

5.1. TREC CAST 2019 RESULTS

BM25 retrieval model and no-reranking on automatic utterances. In experiment #1, no re-writing is applied at all: the system performs poorly due to the lack of contextual information. Results are in line with the baseline provided for this track, as observed in the TREC CAsT 2019 Overview [8]. Experiments #2, #3, and #4 incorporate the rewriter component to the pipeline, which brings forward implied substantives into the text of the utterance, resulting in improved performance. Moreover, the second and third experiments evaluate different CR-based rewriter components, each employing a distinct framework: AllenNLP for the former and LingMess for the latter. Both frameworks can achieve state-of-the-art performance for the CR task, hence their effectiveness is similar in our scenario. We implemented these components performing the same job, given that the AllenNLP framework has been discontinued by its creators and is no longer updated since December 2022. Run #4, which employs T5, is the most effective re-writing strategy for the TREC CAsT 2019 track, achieving twice the performance w.r.t the automatic baseline across all measures. Moreover, the results are comparable to those obtained using the same experimental settings except for employing the utterances manually rewritten by the organizers (#9).

In experiments #5 to #8, the FLC heuristic, which combines the first, previous and current utterances' text, is also applied. This simple heuristic is beneficial, particularly in cases where the rewriter fails to correctly disambiguate pronouns with the relative entity and expand with implied information extracted from within the conversation. In such cases, it allows to introduce missing context information from previous utterances, that would otherwise be lost. Furthermore, the highest improvements are obtained when no re-writing is applied (#5): this is expected, as FLC represent the only mechanism to bring contextual information into the query. In particular, we observe 53.8%, 50.9%, 66.2%, and 59.7% performance uplift on the four metrics considered. Significant improvements can also be achieved when FLC is applied after a rewriter performing CR: for the AllenNLP-based component (#6), the improvements are 22.9%, 18.4%, 20.2%, and 21.1%, while for the Fastcoref-based one (#7) are 17.8%, 18.8%, 19.7%, and 18.5%. Overall, the effectiveness achieved by this combination of components is comparable to the T5 rewriter. Instead, very small improvements are observed with the T5 rewriter (#8): 3.3%, 5.3%, 5.0%, and 2.8% on the four metrics considered. The negligible impact of FLC can be explained as T5 is more effective in enriching the utterance, making simple heuristics such as FLC redundant. Again, results are consistent with the observations made in the TREC CAsT 2019

Overview [8].

Figure 5.1 shows the effectiveness of FLC heuristic in function of the coefficient applied to the First, Last and Current utterances' text. The x and y axes represent the weights given to the first and last utterances, respectively, while $1 - (F + L)$ is the implicit weight assigned to the current utterance. As discussed earlier, the effectiveness of FLC is dependent on which rewriting strategy is employed. Each heatmap shows the nDCG@10 metric achieved considering only utterances whose turn is greater or equal to 3 within its conversation. The optimal values determined from these plots are the coefficients employed for the experiments shown in the previous paragraph (#5 to #8).

When no rewriting is applied, the best performance is achieved using very large values for the first and last coefficients, as shown by Figure 5.1a. Less emphasis on the first utterance is required by the Fastcoref rewriter. Figure 5.1b observes that overemphasizing the coefficients' value can negatively impact the overall effectiveness of the heuristic. Finally, Figure 5.1c shows the negligible impact of FLC when employing the T5 rewriter. It is interesting to note that a significant portion of the plot, especially on the top-left corner, exhibits slightly higher scores w.r.t. considering only the current utterance.

Figure 5.2 displays the impact on nDCG@10 metric of employing the FLC heuristic against considering solely the current utterance. When no rewriting is applied, the FLC heuristic generally improves query effectiveness, as seen in Figure 5.2a. Instead, the result varies whether one of the available rewriter component is used. Figures 5.2b and 5.2c demonstrate that most queries are unaffected by the FLC heuristic, whereas a small number of them experience significant performance differences.

Regardless of the rewriting strategy employed, few queries decrease their effectiveness, such as "31_7" and "67_10", while others show a substantial gain in performance, like "32_6" and "58_7". The former case can be explained as the FLC heuristic introduces non-relevant information to the query, resulting in significant changes to the scores produced by BM25. This phenomenon is most prominent towards the end of the conversation, as topic shifts are commonly employed to drift the trajectory of the dialogue towards related subjects. In the latter case, instead, the rewritten utterance lacks critical information for the success of the query, which is added through the FLC heuristic. Sometimes this is due to the rewriter failing to correctly expand the utterance, sometimes the given utterance has insufficient data to fulfill the user's information need.

5.1. TREC CAST 2019 RESULTS

0.3-	16.17	16.74	17.21	18.17	18.75	19.18	19.57	19.82	20.53	21.08	21.24	21.38	21.88	21.64	21.07	20.94
0.28-	15.81	16.42	17.00	17.74	18.64	19.21	19.51	20.00	20.42	21.21	21.42	21.69	21.83	21.82	21.58	21.48
0.26-	15.35	15.90	16.57	17.24	18.13	18.77	19.38	19.76	20.60	21.12	21.40	21.54	22.03	21.75	21.99	21.88
0.24-	15.25	15.73	16.23	16.94	17.73	18.27	18.95	19.47	20.08	20.59	21.11	21.25	21.67	22.18	21.93	21.98
0.22-	15.05	15.50	15.85	16.48	16.99	17.67	18.48	19.23	19.84	20.29	20.58	21.21	21.44	21.81	22.04	22.14
0.2-	14.65	15.03	15.44	15.99	16.50	17.13	17.85	18.73	19.41	20.01	20.35	20.63	21.12	21.57	21.89	22.03
0.18-	14.20	14.57	15.17	15.73	16.25	16.72	17.35	18.17	19.02	19.67	20.15	20.29	20.82	21.38	21.81	21.95
0.16-	14.10	14.26	14.45	15.07	15.63	16.47	17.05	17.63	18.58	19.28	19.59	19.91	20.45	21.00	21.58	22.09
0.14-	13.87	14.17	14.25	14.79	15.39	16.05	16.44	17.04	17.47	18.45	19.04	19.47	20.09	20.92	21.25	21.89
0.12-	13.73	13.98	14.06	14.39	14.92	15.72	16.24	16.62	17.04	17.68	18.42	18.85	19.88	20.76	20.91	21.36
0.1-	13.59	13.82	14.03	14.33	14.72	15.71	15.84	16.28	16.92	17.20	18.04	18.31	19.29	20.34	20.83	20.93
0.08-	13.10	13.47	13.78	14.08	14.60	15.35	15.45	15.95	16.62	16.90	17.53	17.94	18.76	19.66	20.42	20.68
0.06-	12.72	13.12	13.42	13.84	14.34	15.10	15.43	15.63	16.12	16.85	17.26	17.65	18.42	19.10	19.96	20.61
0.04-	12.86	13.09	13.36	13.77	14.22	14.79	15.12	15.33	15.99	16.54	17.12	17.52	18.05	18.78	19.40	19.84
0.02-	12.73	12.98	13.25	13.43	14.07	14.63	15.04	15.29	15.84	16.28	16.65	17.11	17.73	18.34	19.06	19.73
0.0-	12.49	12.78	13.00	13.13	13.95	14.54	14.73	15.17	15.72	16.06	16.38	16.95	17.62	18.13	18.58	19.39
	0.00	0.02	0.04	0.06	0.08	0.10	0.12	0.14	0.16	0.18	0.20	0.22	0.24	0.26	0.28	0.30

(a) No Re-writing

0.3-	28.66	29.20	29.39	29.64	29.89	30.18	30.52	30.18	29.80	29.98	29.75	28.96	28.73	28.03	27.01	26.09
0.28-	28.40	28.72	28.90	29.36	30.01	30.18	30.36	30.34	30.16	30.16	30.23	29.52	29.02	28.63	27.94	27.20
0.26-	27.96	27.97	28.47	29.19	29.53	29.85	30.05	30.04	30.26	30.31	30.06	29.84	29.25	28.72	28.33	27.88
0.24-	27.95	28.18	28.13	28.51	29.02	29.60	29.83	30.11	30.14	30.18	30.30	29.92	29.53	29.13	28.59	28.24
0.22-	27.66	28.17	28.22	28.54	28.64	29.14	29.53	29.88	30.15	29.92	30.36	30.18	29.77	29.11	28.97	28.43
0.2-	27.34	27.62	28.01	28.18	28.53	28.69	29.19	29.44	29.96	29.87	30.02	30.04	30.18	29.62	29.09	28.57
0.18-	26.98	27.27	27.75	28.09	28.22	28.51	28.73	29.32	29.72	29.89	29.98	29.96	29.98	30.00	29.79	29.09
0.16-	27.13	27.08	27.28	27.54	27.93	28.15	28.39	28.79	29.32	29.38	29.35	29.45	29.92	29.93	29.90	29.81
0.14-	26.79	27.08	27.05	27.33	27.64	27.91	28.06	28.33	28.40	28.79	29.10	29.08	29.45	29.62	29.83	30.04
0.12-	26.74	27.14	27.06	27.04	27.38	27.64	27.78	27.99	28.05	28.21	28.72	28.65	28.98	29.30	29.53	29.58
0.1-	26.73	26.92	27.02	27.25	27.17	27.63	27.60	27.87	27.99	27.90	28.36	28.33	28.76	29.00	29.00	29.16
0.08-	26.38	26.59	26.73	26.85	27.13	27.46	27.65	27.73	27.84	27.88	28.08	28.14	28.15	28.70	28.81	28.74
0.06-	26.13	26.32	26.44	26.75	26.90	27.40	27.72	27.85	27.83	27.75	27.83	27.94	28.01	28.26	28.76	28.61
0.04-	26.26	26.12	26.22	26.63	26.63	27.06	27.44	27.58	27.73	27.70	27.71	27.77	27.96	27.85	28.30	28.59
0.02-	25.96	26.08	26.15	26.24	26.47	26.76	27.22	27.39	27.51	27.53	27.57	27.45	27.78	27.67	27.98	28.25
0.0-	25.70	25.88	25.85	25.78	26.28	26.64	26.73	27.09	27.35	27.26	27.30	27.39	27.59	27.66	27.61	27.76
	0.00	0.02	0.04	0.06	0.08	0.10	0.12	0.14	0.16	0.18	0.20	0.22	0.24	0.26	0.28	0.30

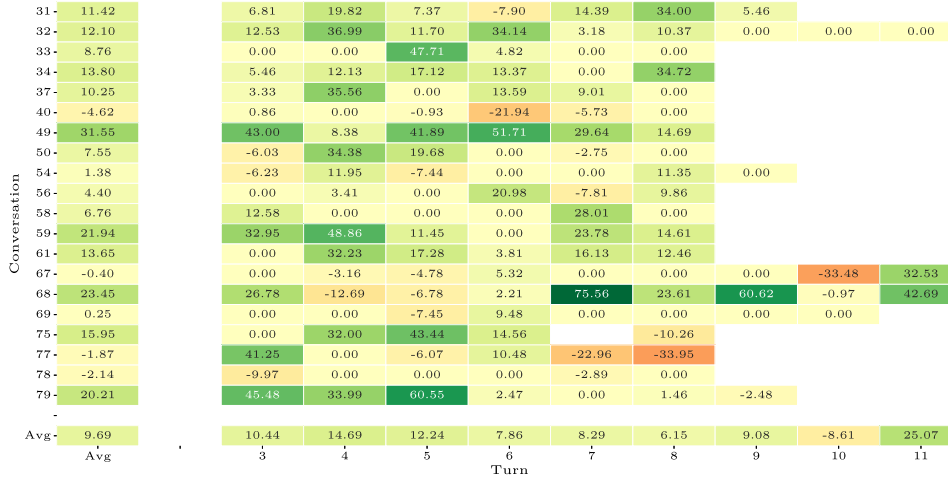
(b) Fastcoref Co-reference Resolution Re-writing

0.3-	33.71	33.68	33.51	33.28	33.21	32.93	32.97	32.36	31.56	30.94	30.36	29.61	28.81	27.88	27.01	25.66
0.28-	33.98	34.07	33.42	33.50	33.52	33.26	33.00	32.53	31.94	31.33	30.65	30.28	29.08	28.62	27.61	26.72
0.26-	33.84	33.88	33.84	33.85	33.53	33.40	32.94	32.73	32.22	32.01	31.17	30.87	29.90	29.17	28.31	27.34
0.24-	33.78	33.88	33.67	33.91	33.63	33.66	33.38	33.25	32.85	32.09	31.56	31.15	30.70	30.03	29.10	28.26
0.22-	33.73	33.80	33.88	33.84	33.58	33.49	33.43	33.48	33.33	32.64	31.91	31.71	31.16	30.45	29.61	29.00
0.2-	33.88	33.70	33.69	33.48	33.50	33.51	33.38	33.27	33.39	32.89	32.52	32.05	31.44	31.03	30.32	29.51
0.18-	33.50	33.52	33.90	33.56	33.35	33.23	33.45	33.24	33.22	33.16	32.76	32.41	31.87	31.53	30.91	30.41
0.16-	33.63	33.37	33.40	33.35	33.47	33.33	33.34	33.16	33.23	32.89	32.92	32.54	32.46	32.12	31.59	30.82
0.14-	33.32	33.52	33.48	33.14	33.27	33.39	33.26	33.22	33.18	33.11	32.73	32.72	32.51	32.12	31.88	31.47
0.12-	33.45	33.56	33.53	33.42	33.44	33.50	33.42	33.31	33.09	32.93	32.78	32.43	32.39	32.46	32.04	31.59
0.1-	33.61	33.31	33.22	33.41	33.29	33.44	33.28	33.28	33.07	33.00	32.81	32.57	32.32	32.21	32.16	31.66
0.08-	33.27	33.09	33.13	33.16	33.14	33.11	33.12	33.18	33.02	32.99	33.00	32.79	32.27	32.14	31.93	31.69
0.06-	33.01	33.08	33.20	33.27	33.13	33.13	33.14	33.10	33.11	33.06	32.76	32.75	32.65	32.22	32.11	31.71
0.04-	33.11	32.99	33.02	33.12	33.00	32.90	33.01	33.06	32.90	32.75	32.82	32.56	32.48	32.38	32.31	31.91
0.02-	33.21	33.00	32.84	33.00	33.03	32.90	32.88	33.00	32.83	32.76	32.57	32.49	32.29	32.07	32.30	32.17
0.0-	32.94	32.96	32.83	32.70	32.83	32.78	32.66	32.76	32.80	32.55	32.35	32.25	32.47	32.09	31.81	31.75
	0.00	0.02	0.04	0.06	0.08	0.10	0.12	0.14	0.16	0.18	0.20	0.22	0.24	0.26	0.28	0.30

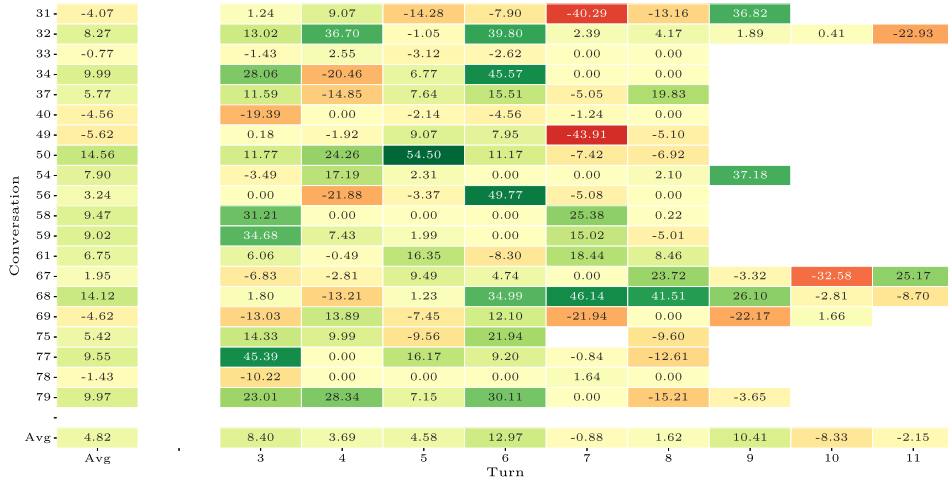
(c) T5 Re-writing

Figure 5.1: Evaluation of FLC heuristic paired with three different rewriting strategy. The scoring function is BM25 and no re-ranking is employed. Each column and row represent the coefficient applied to the First and Last utterances, respectively. Instead, the coefficient for Current utterance is given by $1 - (F + L)$. The evaluation metric reported is nDCG@10, considering only utterances those turn is ≥ 3 within its conversation. 62

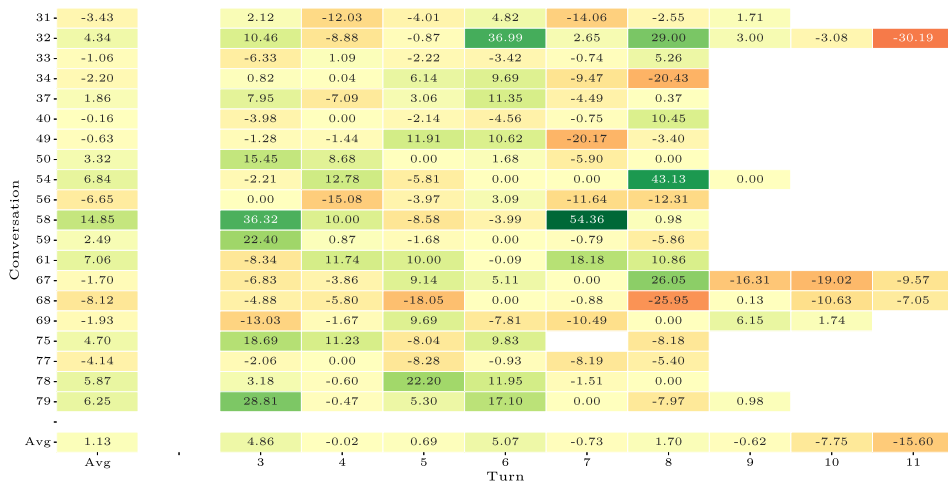
CHAPTER 5. EXPERIMENTAL RESULTS



(a) No Re-writing



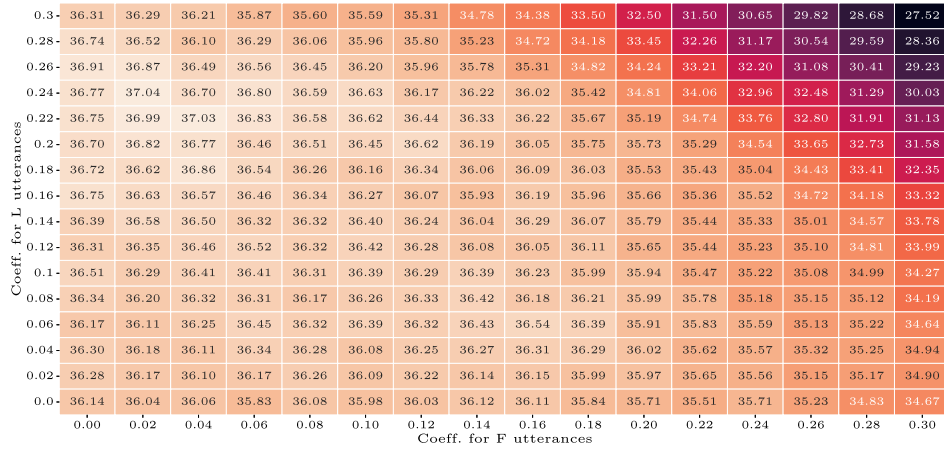
(b) Fastcoref Co-reference Resolution Re-writing



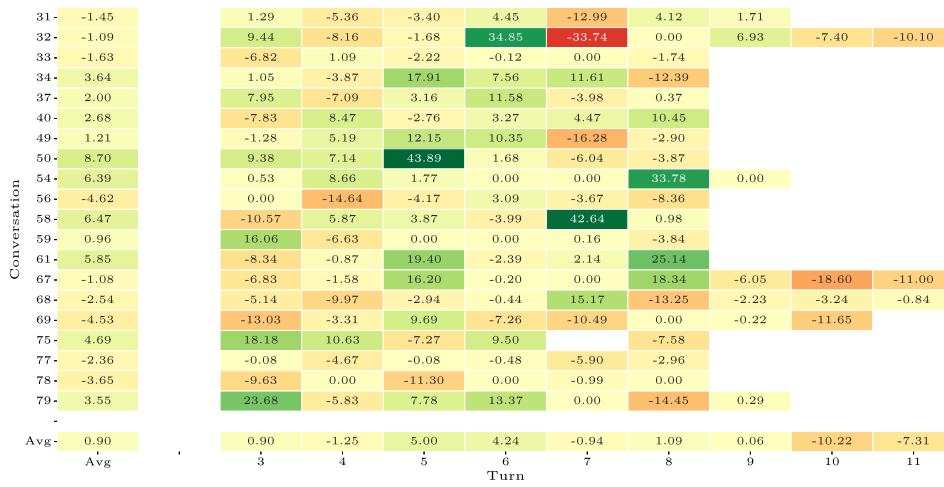
(c) T5 Re-writing

Figure 5.2: Differences in nDCG@10 evaluation metric across all queries, between the run employing FLC w.r.t. considering only the current utterance. Three different rewriting strategies are employed, one for each plot, the scoring function is BM25, and no re-ranking is used. Only the utterances those turn is ≥ 3 within its conversation are considered.

5.1. TREC CAST 2019 RESULTS



(a) Heatmap with FLC coefficients.



(b) Differences in nDCG@10 metric between FLC vs. current utterance.

Figure 5.3: Evaluation of the same experiments showed on Figures 5.1 and 5.2 conducted on the “manual” utterances, which were manually rewritten from the organizers of TREC CAST 2019.

Table 5.5: Optimal parameters to employ with FLC heuristic for maximizing nDCG@10 metric, determined with various rewriting strategies and type of utterances, on TREC CAsT 2019 benchmark.

Scor. Fun.	Topics	Rewriter	F Coeff.	L Coeff.	C Coeff.
BM25	Auto.	—	0.26	0.24	0.50
		A-CR	0.20	0.28	0.52
		F-CR	0.12	0.30	0.58
		T5	0.02	0.28	0.70
	Man.	—	0.02	0.24	0.74
	DIR.	Auto.	—	0.30	0.18
A-CR			0.22	0.20	0.58
F-CR			0.24	0.18	0.58
T5			0.06	0.22	0.72
Man.		—	0.06	0.02	0.92

Figure 5.3 shows the experiments conducted using the same settings on the “manual” utterances, which were manually rewritten by the organizers of TREC CAsT 2019. Both plots closely resemble the equivalent ones generated from automatic utterances rewritten using the T5 rewriter. Focusing on Figure 5.3b, the same observations could be done as those described in the previous paragraph. We can conclude that, indeed, a few of the provided utterances are poor natural language formulations for the information need they represent. For instance, conversation “32” focuses on sharks, considering their different species and sizes, their endangered status, then focuses on asking details about mako sharks. The utterance for turn 6 exhibits significant improvements from FLC heuristic. Its rewritten text from the organizers, “What about for great whites?”, lacks any reference to the user’s specific interests. The previous utterance, “What’s the biggest shark ever caught?”, contains the required information.

The FLC heuristic is beneficial for many different retrieval models and scoring function. In this section, we showed the data about BM25, but similar results can be found using Dirichlet LM. Table 5.5 shows the optimal parameters to use with FLC, as function of the rewriting strategy and type of utterances, which are employed on our experiments throughout the entire chapter. However, due to time constraints, we did not perform a comprehensive investigation about its performance employing BERT and SPLADE as first-stage retrieval models. Preliminary experiments suggest that the heuristic may be beneficial for BERT, while is detrimental to the overall performance for SPLADE. In the latter case,

Table 5.6: List of experiments conducted on TREC CAsT 2019 benchmark, testing different ranking functions, detailed in Section 5.1.2. The evaluation measures reported are R@100, MRR and nDCG with cutoffs 3 and 10.

	Topics	Rew.	Query Gen.	Searcher	Rec.	MRR	nDCG@3	nDCG@10
1	Auto.	T5	C	BM25	42.8	64.0	33.9	32.1
2			FLC	BM25	44.2	67.4	35.6	33.0
3			C	DIR.	45.5	65.2	35.7	34.5
4			FLC	DIR.	47.2	66.8	35.9	35.1
5			C	BERT	43.2	52.3	30.4	33.1
6			C	SPLADE	51.5	79.9	52.3	50.1
7	Man.	—	C	BM25	47.8	66.7	35.4	34.5
8			FLC	BM25	49.5	70.2	36.6	35.2
9			C	DIR.	49.6	68.0	37.6	37.0
10			FLC	DIR.	50.0	68.8	37.9	37.7
11			C	BERT	46.4	54.3	32.8	35.5
12			C	SPLADE	54.9	84.3	56.6	53.5

this could be attributed to the indexing process, which introduces additional terms not present in the text, thereby increasing the likelihood of relevant documents matching the original query. Hence the positive impact of FLC heuristic is negligible, while, on the other hand, the inclusion of irrelevant information to the query makes it less effective.

5.1.2 FIRST-STAGE RETRIEVAL

Table 5.6 shows the results obtained testing different retrieval models and scoring functions. In this section, we evaluate both manual and automatic utterances rewritten using the T5 rewriter, which has been proved in Section 5.1.1 to be the most effective rewriting strategy on the TREC CAsT 2019 settings.

The experiments showed in the top half of Table 5.6 employ automatic utterances rewritten using T5. The first two focus on BM25 as scoring function, with #2 also using the FLC heuristic. The same settings are utilized for the next couple of experiments, except the scoring function is now Dirichlet LM. The Dirichlet LM slightly boost both Recall and nDCG@10 performance metrics compared to BM25. The FLC variant is also beneficial for this scoring function, as demonstrated by experiment #4.

Experiment #5 employs a first-stage dense retrieval approach using a BERT model fine-tuned specifically for IR. The performance obtained is comparable

to standard lexical BoW models such as BM25 and Dirichlet LM, but this approach is less effective when compared against methods that also employ a re-ranking phase, as will be detailed in Section 5.1.3. In fact, the same BERT model can be used for re-ranking, which remarkably improves precision at small cut-offs. However, the dense retrieval approach doesn't significantly benefit from this additional phase, as the performance gain is much smaller. This is due to both the first-stage and re-ranking models (which must be different) evaluating mainly the same semantic features of documents' text. Moreover, the larger storage space required to store the index together with the higher computational resources and time needed make this approach the least effective overall for this particular setting. Test #6 demonstrates that SPLADE reaches the best performance measures, outperforming BM25 (#1) by 20.3% in terms of recall, 54.3% for nDCG@3 and 47.8% for nDCG@10 when T5 rewriter is used. The performance differences against classical lexical models will be much lower when they are paired with a re-ranking phase, as will be discussed in Section 5.1.3.

The same conclusions can be drawn from experiments considering the manual utterances.

5.1.3 RE-RANKING

Table 5.7 demonstrate that performing an additional re-ranking step using a BERT model can further improve performance for all experimental settings, except when SPLADE is used. A comparison between experiment #1 and #2 shows that, despite the absence of any form of rewriting, the performance is significantly improved by using BERT as reranker, thus alleviating the issues related to the "context representation" introduced by the CS setting. Besides, the results of experiment #2 are similar to the average performance of systems submitted at TREC CAsT 2019 [8]. Tests #3 and #4 shows that the FLC heuristic, here employed as the solely technique to resolve context, together with BERT reranking can slightly outperform state-of-the-art rewriting methods such as T5 without re-ranking (#5), when considering the nDCG metric at small cutoffs.

Experiments #6, #8, #10, and #12 demonstrate the effectiveness of BERT-based re-ranking for classical lexical BoW models such as BM25 and Dirichlet LM, resulting in an average improvement of 21.5% in terms of MRR, 44.2 % for nDCG@3, and 36.9 % for nDCG@10. Furthermore, these experiments achieve comparable performance to SPLADE. For instance, test #10, using Dirichlet LM with

Table 5.7: List of experiments conducted on TREC CAsT 2019 benchmark, applying re-ranking using a BERT model to different settings, detailed in Section 5.1.3. The evaluation measures reported are R@100, MRR and nDCG with cut-offs 3 and 10.

	Topics	Rew.	Searcher	Rer.	Rec.	MRR	nDCG@3	nDCG@10
1	Auto.	—	BM25 _C	—	19.9	32.0	14.2	14.4
2			BM25 _C	BERT	19.9	48.0	28.4	24.9
3			BM25 _{FLC}	—	30.6	48.3	23.6	23.0
4			BM25 _{FLC}	BERT	30.6	60.2	36.4	33.1
5	Auto.	T5	BM25 _C	—	42.8	64.0	33.9	32.1
6			BM25 _C	BERT	42.8	79.3	50.4	45.2
7			BM25 _{FLC}	—	44.2	67.4	35.6	33.0
8			BM25 _{FLC}	BERT	44.2	78.3	50.4	46.0
9			DIR. _C	—	45.5	65.2	35.7	34.5
10			DIR. _C	BERT	45.5	81.5	51.9	46.4
11			DIR. _{FLC}	—	47.2	66.8	35.9	35.1
12			DIR. _{FLC}	BERT	47.2	80.7	50.7	46.6
13			SPLADE _C	—	51.5	79.9	52.3	50.1
14			SPLADE _C	BERT	51.5	73.6	45.7	45.3

BERT, slightly outperforms SPLADE (#13) by 2.0 % for MRR, but is less effective for 11.7 %, 0.8 %, and 7.4 % in terms of Recall, nDCG@3, and nDCG@10, respectively. The use of FLC heuristic during first-stage retrieval, when both T5 rewriter and the additional re-ranking phase are employed, is beneficial for both Recall and nDCG@10, but is detrimental for MRR and nDCG@3. Overall, the effectiveness of FLC heuristic when paired with state-of-the-art rewriting and re-ranking methods is uncertain. Instead, experiment #14 shows that performing re-ranking is detrimental to the overall performance when SPLADE is used as the first-stage retrieval model. This is expected, since SPLADE has been specifically developed to achieve state-of-the-art performance without requiring additional re-ranking.

Figure 5.4 shows the differences in nDCG@10 metric for each query between experiments employing re-ranking against the same without re-ranking. The first plot, Figure 5.4a, shows the impact of BERT re-ranking using BM25 scoring function during first-stage retrieval, by comparing experiments #6 and #5. For the second plot, Figure 5.4b, Dirichlet LM is utilized (experiments #10 and #9).

Most queries are considerably improved by this additional phase, except for

31	23.54	44.15	40.93	37.17	29.55	5.15	20.48	12.77	-0.20	21.85		
32	8.02	23.05	-12.68	49.53	18.91	-2.65	5.88	-0.96	11.30	6.00	16.83	-26.98
33	16.32	31.92	36.20	-12.79	4.64	32.52	24.63	13.61	-0.15			
34	-1.91	16.51	-10.58	-19.84	-3.62	2.77	19.39	19.62	-39.54			
37	11.38	23.30	-1.89	14.07	-5.96	43.23	2.48	-5.89	21.70			
40	14.26	46.47	13.34	35.80	0.00	0.39	17.10	20.19	-19.22			
49	5.60	26.92	7.13	15.81	34.79	-4.27	-40.02	-18.65	23.11			
50	23.57	18.36	-4.34	40.38	18.11	48.47	57.69	9.91	0.00			
54	3.62	-4.46	-28.10	-6.70	6.33	-9.03	0.00	0.00	74.55	0.00		
56	0.49	14.08	10.16	19.18	-2.01	-6.16	8.60	-21.08	-18.83			
58	17.44	16.21	9.92	37.43	8.06	14.54	-16.76	64.32	5.83			
59	17.75	18.63	25.34	42.01	-2.16	29.20	0.00	4.73	24.27			
61	13.34	-7.40	0.00	30.15	-1.41	14.74	41.83	17.13	11.70			
67	12.13	13.14	37.05	32.33	6.37	13.24	30.06	0.00	32.35	-6.02	20.85	-45.07
68	13.09	11.42	2.43	38.51	38.36	13.89	0.00	-0.26	10.35	-4.33	23.00	10.66
69	13.93	7.15	5.24	23.45	14.16	4.36	23.85	20.40	32.54	8.17	0.00	
75	27.25	22.90	0.68	47.87	37.88	32.46	26.76		22.21			
77	5.74	33.34	0.00	14.33	0.00	5.12	38.27	-43.85	-1.28			
78	18.24	24.38	23.84	39.76	13.89	33.75	0.00	10.30	0.00			
79	15.35	12.84	-30.64	51.08	13.79	24.86	20.37	0.00	30.39	15.43		
Avg	12.81	19.65	6.20	26.48	11.48	14.83	14.03	5.38	11.05	5.87	15.17	-20.77
Avg		1	2	3	4	5	6	7	8	9	10	11

(a) BM25

31	12.74	3.60	-11.63	-15.62	18.32	19.25	41.92	56.92	-13.30	15.16		
32	14.60	23.21	-0.37	65.49	42.75	4.99	4.97	-13.89	11.30	6.00	26.16	-10.06
33	30.11	43.95	52.04	8.55	18.45	28.35	49.08	30.87	9.63			
34	6.21	21.40	-11.49	9.26	7.50	22.84	9.69	7.58	-17.08			
37	15.44	41.11	3.02	29.94	13.12	37.56	8.74	-2.28	-7.67			
40	16.25	22.04	17.78	42.68	32.06	2.37	3.14	36.59	-26.69			
49	6.07	40.71	25.51	-1.11	11.35	0.79	-44.97	-5.96	22.27			
50	24.48	56.51	7.93	50.06	-7.10	36.18	39.74	12.52	0.00			
54	11.55	9.56	41.53	13.87	-0.97	2.04	0.00	0.00	14.55	23.32		
56	1.27	36.06	-0.72	7.09	-8.10	-8.93	7.83	-19.26	-3.77			
58	6.31	20.33	-5.14	-2.91	-13.73	14.58	-29.39	52.34	14.41			
59	7.53	13.29	7.59	34.11	4.73	0.00	0.00	-8.62	9.14			
61	9.07	-18.19	-5.84	17.76	35.65	0.43	5.80	28.06	8.92			
67	0.80	36.88	-18.82	32.46	5.31	-29.51	15.89	0.00	3.83	-33.66	19.86	-23.48
68	14.67	22.29	2.43	55.80	30.19	-0.08	0.00	3.13	17.81	-8.61	22.15	16.21
69	12.27	22.01	14.19	15.01	-12.45	2.01	21.25	3.16	39.71	17.84	0.00	
75	34.61	27.61	53.87	23.88	19.09	44.39	40.09		33.30			
77	3.37	32.38	0.00	19.39	0.00	2.44	-2.11	-22.43	-2.70			
78	14.06	18.70	19.73	27.53	16.59	-1.30	0.00	31.25	0.00			
79	3.20	-7.39	-25.59	19.13	27.61	21.58	21.34	-13.34	11.87	-26.37		
Avg	11.93	23.30	8.30	22.62	12.02	10.00	9.65	9.30	6.28	-0.90	17.04	-5.77
Avg		1	2	3	4	5	6	7	8	9	10	11

(b) Dirichlet LM

Figure 5.4: Differences in nDCG@10 evaluation metric across all queries, between the run employing reranking with BERT w.r.t. no reranking. Both experiments are based on T5-rewritten automatic utterances, but two different ranking function are employed, one for each plot: BM25 and Dirichlet LM.

5.1. TREC CAST 2019 RESULTS

a small subset, such as “34_8”, “49_6”, “56_6”, and “79_2”, which shows in both plots a significant decrease in performance. A quick examination of these utterances reveals that most of them are related to named entities: Bronze Age collapse, Amazon Prime Video, Darwin, and Auguste Comte, respectively.

FLC HEURISTIC AND RUN FUSION

The FLC heuristic can also be applied to the query representation used by BERT-based re-ranking. Due to time constraints, only the automatic utterances rewritten with T5 model has been examined. Figure 5.5a shows the variation of nDCG@10 metric as function of the coefficients used for the first and last utterances. The results indicate that this simple heuristic is never beneficial when employed for re-ranking.

Another approach to further improve the effectiveness of re-ranking is to perform run fusion between the ranked list produced by the searcher during first-stage retrieval with that generated by the reranker. Figure 5.5b shows the average nDCG@10 metric across all queries when the rankings are merged using our custom Reciprocal Rank technique, while varying the linear coefficient. The plot exhibits small increments around values between 0.7 and 0.9 for the linear coefficient, with a maximum gain of 2.6 %. The impact of this approach on the overall effectiveness is negligible.

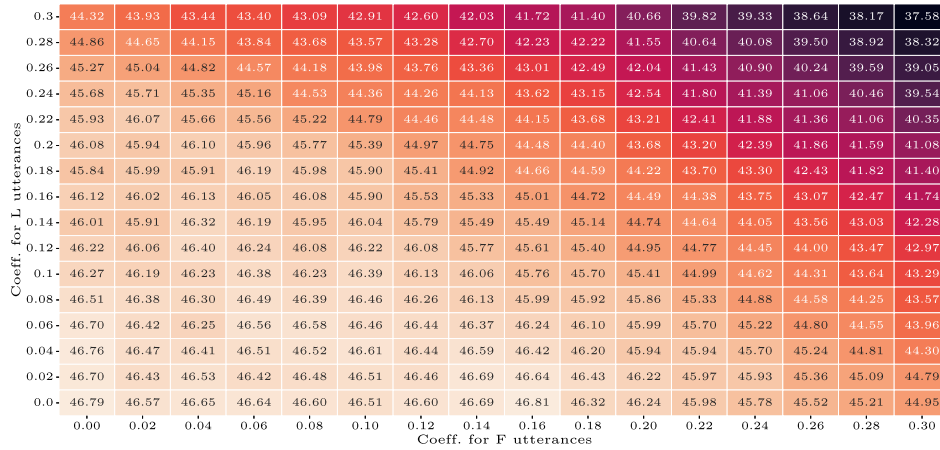
5.1.4 COMPARISON WITH BEST RUNS OF TREC CAST 2019

Table 5.8 shows the experiments conducted on TREC CAST 2019 against the best runs across original submissions, for both automatic and manual utterances. We focus our testing solely on those pipelines that best performed throughout the experiments detailed in the previous sections.

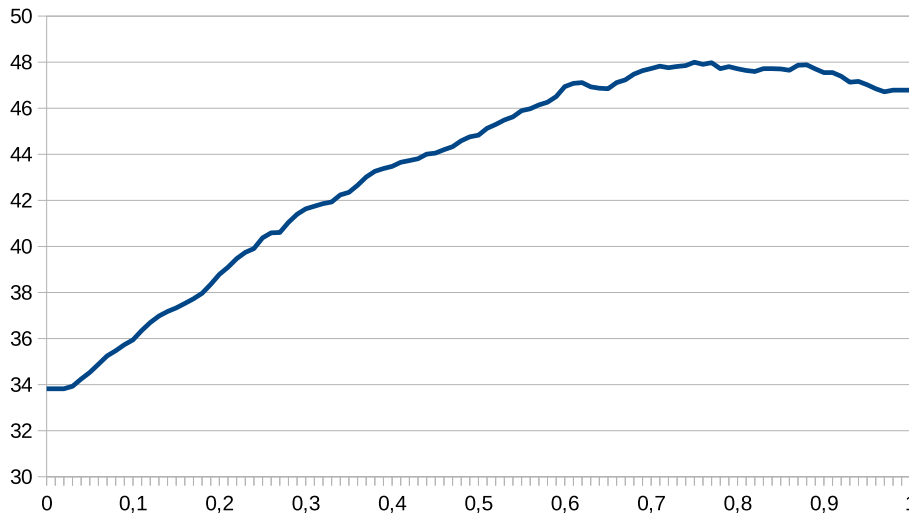
The first part of Table 5.8 focus on automatic utterances, which in our experiments have been rewritten using our T5-based rewriter. In Section 5.1.1, it has been proved to be the most effective rewriting strategy among whose implemented in DECAF. The FLC heuristic is not employed, since in Section 5.1.3 it has not emerged as clearly improving the performance across all four evaluation metrics considered. All three of our experiments outperform the Best Auto-

²The Best Automatic Run is “CFDA_CLIP_RUN7” from Group “CFDA_CLIP”.

³The Best Manual Run is “humanbert” from Group “ATeam”.



(a) Heatmap with FLC coefficients.



(b) Fusion between the ranked lists produced by the Searcher (left) and the Reranker (right).

Figure 5.5: Evaluation of two different strategies for improving re-ranking. Figure 5.5a evaluates the optimal parameters for the FLC heuristic, when applied to the query representation used for re-ranking with BERT, employed to T5-rewritten automatic utterances. Figure 5.5b shows how the score changes when varying the linear combination parameter in our custom Reciprocal Rank run fusion technique, employed to merge the ranked lists produced by the Searcher (left) and the Reranker (right). The performance measure considered is nDCG@10 in both plots.

5.1. TREC CAST 2019 RESULTS

Table 5.8: Comparison between our best-performing experiments against the best runs across original submissions, for both automatic and manual utterances, on TREC CAsT 2019 benchmark. The evaluation measures reported are R@100, MRR and nDCG with cutoffs 3 and 10.

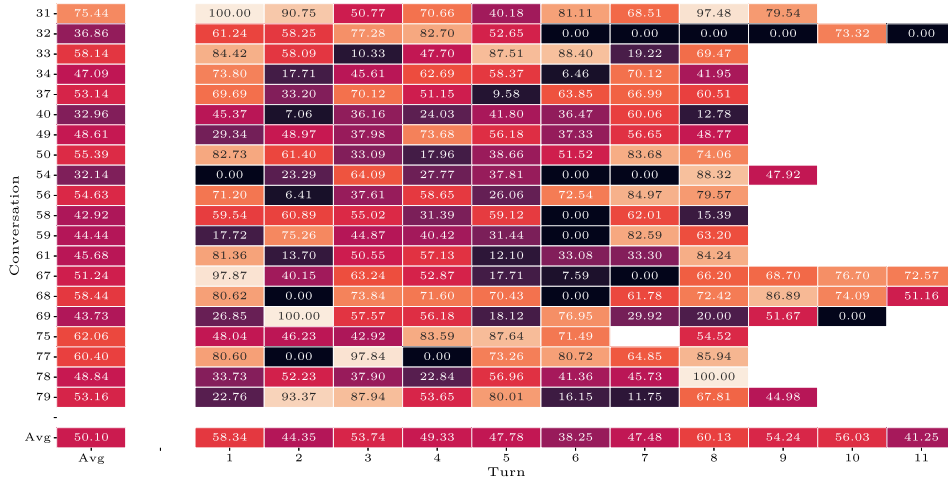
	Topics	Rew.	Searcher	Rer.	Rec.	MRR	nDCG@3	nDCG@10
1	Auto.	T5	BM25 _c	BERT	42.8	79.3	50.4	45.2
2			DIR. _c	BERT	45.5	81.5	51.9	46.4
3			SPLADE _c	—	51.5	79.9	52.3	50.1
Best Automatic Run ² at TREC CAsT 2019					41.2	71.1	42.4	40.6
4	Man.	—	BM25 _c	BERT	47.8	82.5	54.4	48.2
5			DIR. _c	BERT	49.6	85.4	56.1	49.8
6			SPLADE _c	—	54.9	84.3	56.6	53.5
Best Manual Run ³ at TREC CAsT 2019					56.7	88.4	59.3	60.6

matic (BA) run across the original submissions of TREC CAsT 2019, with average differences of 13.1 %, 12.8 %, 21.5 %, and 16.3 % for Recall, MRR, nDCG@3, and nDCG@10 metrics, respectively. The SPLADE model (experiment #3) achieves the best overall performance, with remarkable improvements for both Recall and nDCG@10 metrics w.r.t. all other runs.

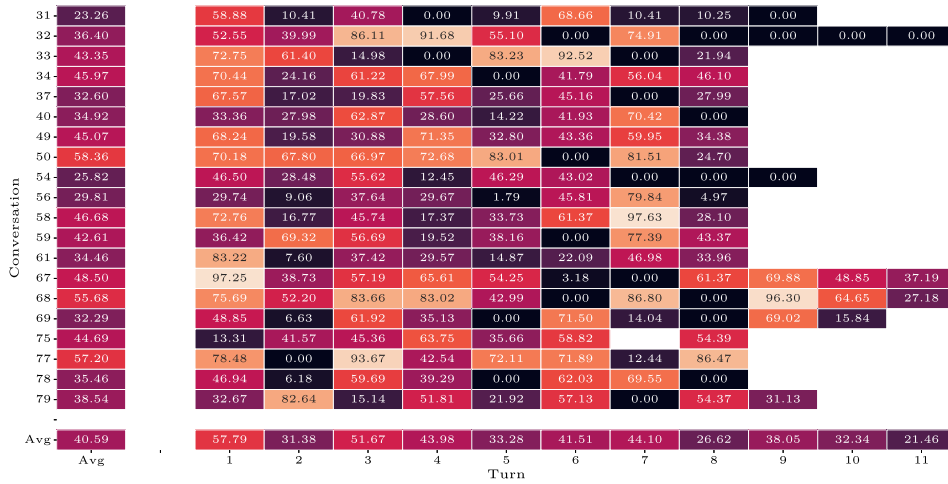
Figure 5.6 shows the nDCG@10 metric across each query, for both the SPLADE and the BA run. These plots exhibit the superior rewriting strategy employed in our experiments, as the average performance across the first query of each conversation is approximately the same.

We now examine the second part of Table 5.8, where the manually rewritten utterances are used. Notice that, the performance observed on this kind of utterance represents an upper bound on the performance that can be achieved by retrieval systems, since in a real-case scenario such utterances would not be available. The performance differences between automatic and manual utterances are mostly independent of the retrieval model utilized, with average differences of 9.1% for recall, 4.8% for MRR, 8.1% for nDCG@3 and 6.9% for nDCG@10. These experiments demonstrate that T5-based query rewriting methods are very effective on the TREC CAsT 2019 dataset. The best performance is achieved again using the SPLADE model (experiment #6), similar to the results observed for the automatic runs. When comparing our best manual run to the Best Manual (BM) run among the original submissions, we observe slightly lower scores, particularly for the nDCG@10 measure with a difference of 13.3%. Figure 5.7

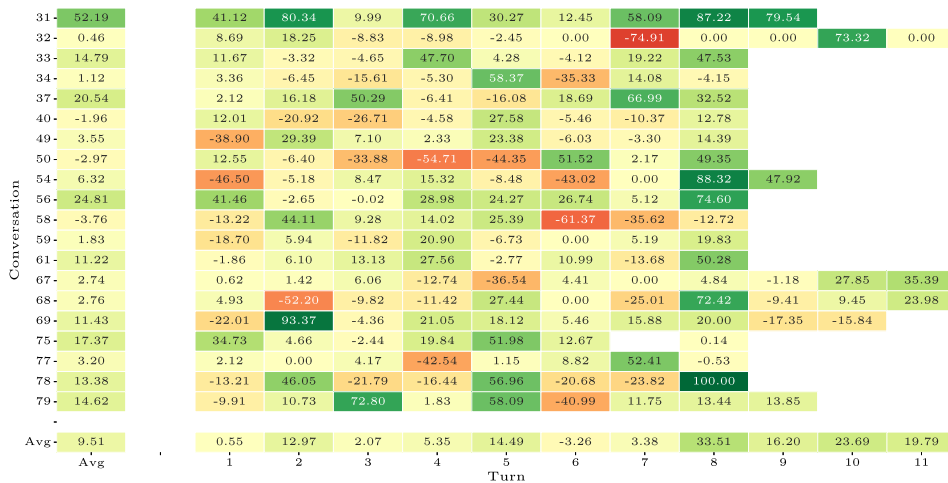
CHAPTER 5. EXPERIMENTAL RESULTS



(a) Automatic Run using SPLADE



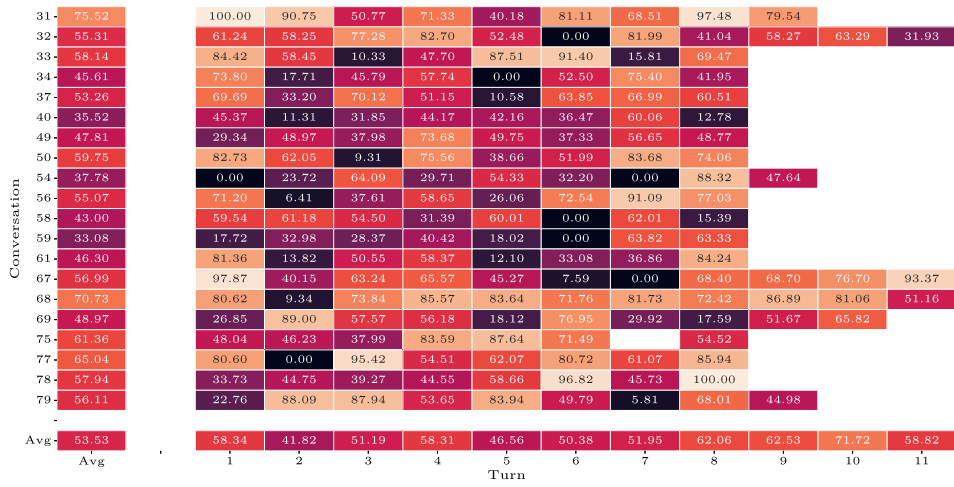
(b) Best Automatic Run



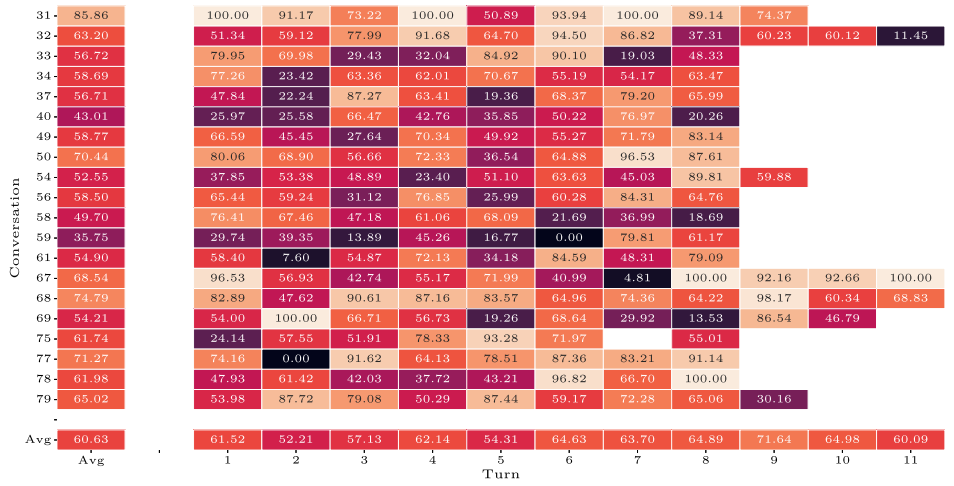
(c) Comparison between SPLADE and Best Automatic Run

Figure 5.6: Evaluation of the nDCG@10 metric across all queries, for both our best experiment on automatic run, which employs SPLADE, and for the Best Automatic run on TREC CAst 2019 dataset.

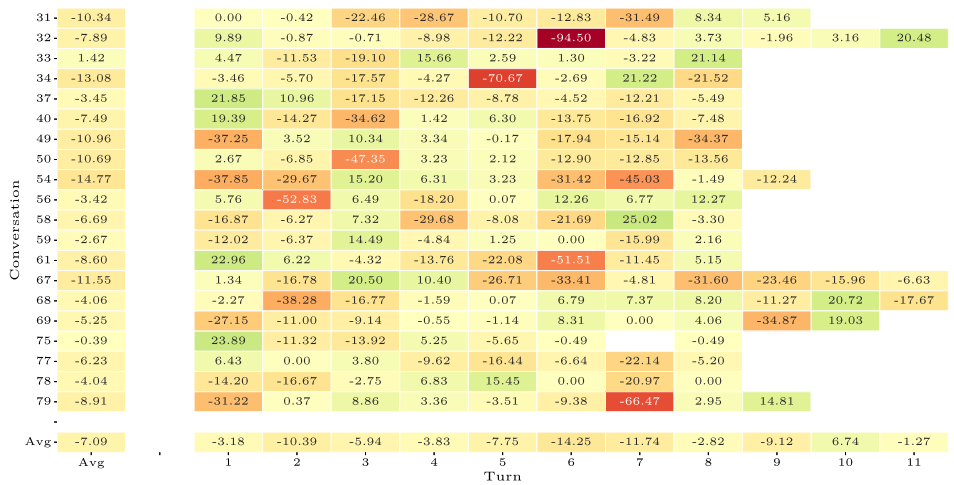
5.1. TREC CAST 2019 RESULTS



(a) Manual Run using SPLADE



(b) Best Manual Run



(c) Comparison between SPLADE and Best Manual Run

Figure 5.7: Evaluation of the nDCG@10 metric across all queries, for both our best experiment on manual run, which employs SPLADE, and for the Best Manual run on TREC CAst 2019 dataset.

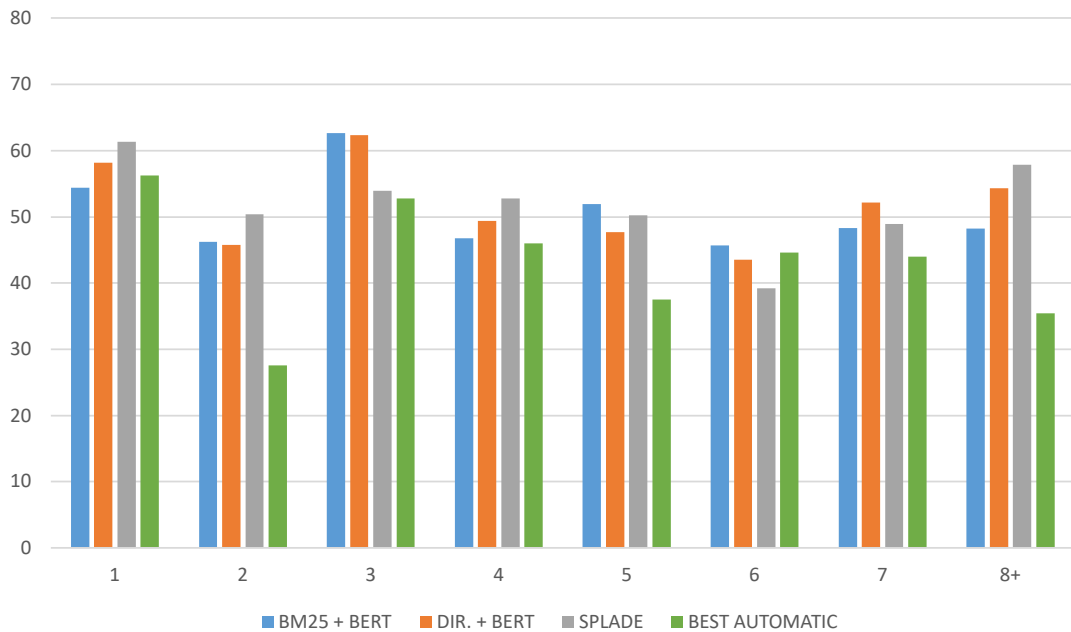
shows the nDCG@10 metric across each query, for both the SPLADE and the BM run. We can see that fewer queries achieves very low results in the BM run w.r.t. our experiment, resulting in higher overall results.

Figure 5.8 shows the average nDCG@3 metric at varying turn depths for both automatic and manual runs on TREC CAsT 2019. In these plots we compared four runs: the first one is BM25 with BERT re-ranking (blue bar, on the left), then Dirichlet LM with BERT re-ranking (red bar, in the center-left), the third one is SPLADE (yellow bar, in the center-right), and the Best of the original submissions (green bar, on the right). Due to smaller sample size, we decided to aggregate all turns with depth greater or equal to 8 together with the same "8+" label. Figure 5.8a exhibits that scores remain almost constant across the whole conversation, thanks to the remarkable effectiveness of T5 rewriter on this dataset. It is worth noting that our system does not suffer any significant drip on depth 2, unlike all original submissions. The same trend can be observed in Figure 5.8b for manual runs, just with slightly higher scores.

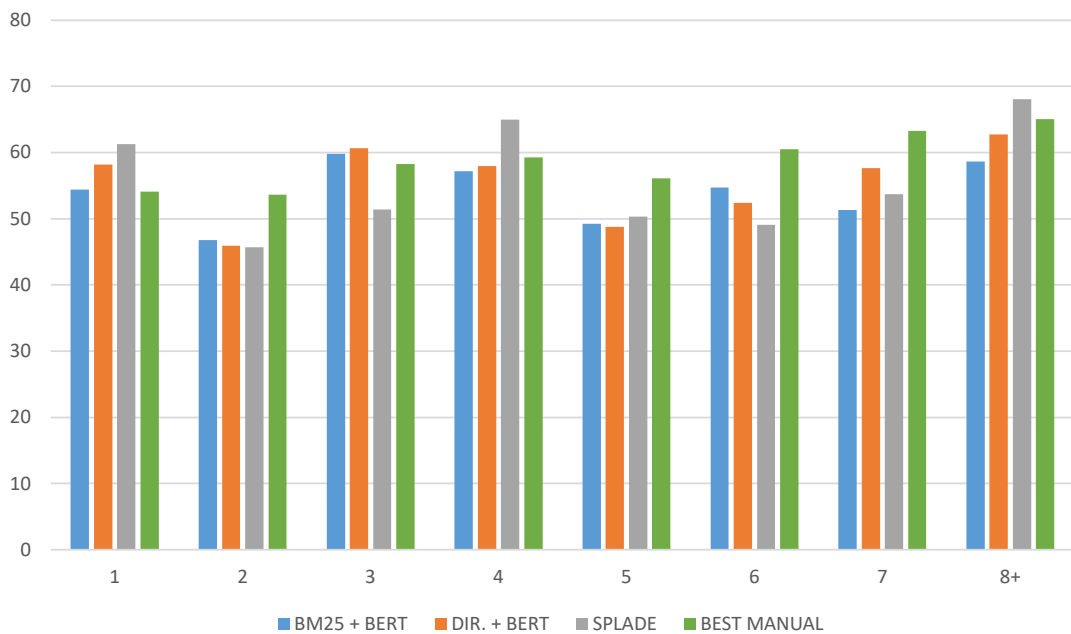
5.2 TREC CAsT 2020 RESULTS

Table 5.9 shows the experimental results on TREC CAsT 2020. In this Section, we consider only the configurations that best performed on TREC CAsT 2019 and evaluate them on the second edition of the track. For automatic runs, we test the three searchers paired with the T5 rewriter. When considering the performance after first-stage retrieval, BM25 (#1) is the lowest performing method, followed by Dirichlet LM (#3) and BERT-based dense retrieval (#5). As observed for TREC CAsT 2019, after the re-ranking phase, the dense retrieval approach (experiment #5) gives the lowest overall scores, followed by Dirichlet LM (#4) and BM25 (#2) with BERT re-ranking, and SPLADE (#6). However, its Recall is much higher than both experiments employing BM25 and Dirichlet LM, with an increment of 36.5 % and 41.8 % respectively. Finally, following what was noticed on TREC CAsT 2019, SPLADE appears to be overall the best approach among those implemented. The patterns remain substantially the same when we switch from automatic to manual runs, likely due to the fact that T5 performs particularly well in resolving the utterances. The automatic baseline ranks third for all measures, while the manual baseline slightly outperforms our best run. Notice that both baselines are the worst-performing run in their respective categories when considering the Recall measure. For reference, we also included

5.2. TREC CAST 2020 RESULTS



(a) Automatic Runs



(b) Manual Runs

Figure 5.8: Average nDCG@3 metric at varying conversation depths, on TREC CaST 2019 dataset. In the top plot, there are automatic runs, while the other displays manual runs. The considered runs are: BM25 with BERT (blue bar, on the left), Dirichlet LM with BERT (orange bar, in the center-left), SPLADE (grey bar, in the center-right), and the Best run across original submissions (green bar, on the right).

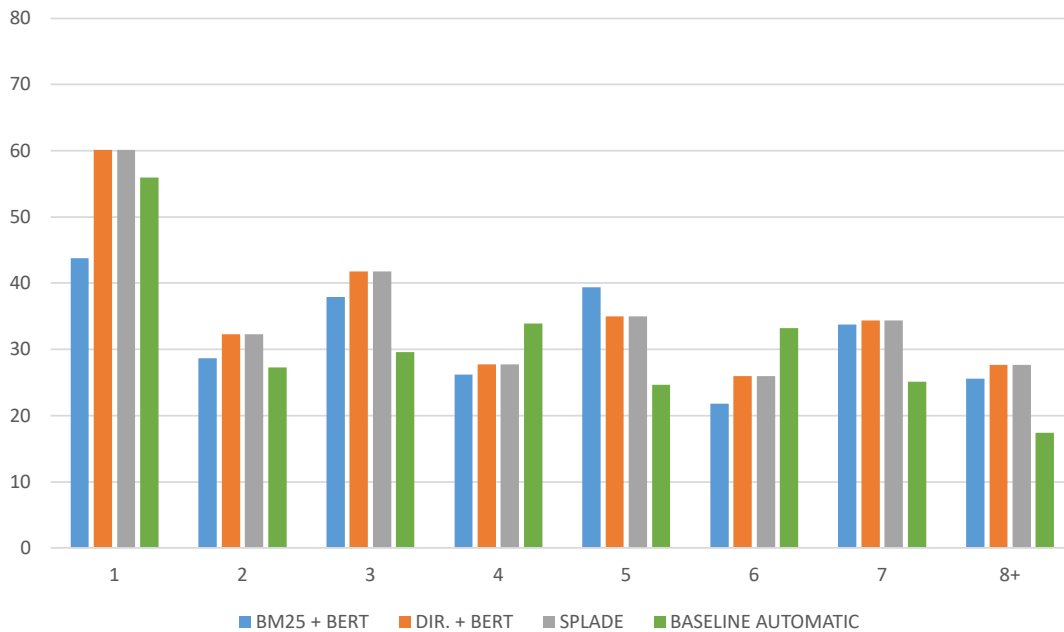
Table 5.9: List of all experiments conducted on TREC CAsT 2020 benchmark. The evaluation measures reported are R@100, MRR and nDCG with cutoffs 3 and 10. We do not report Recall@100 and nDCG@10 for the best TREC CAsT 2020 runs, since they were not computed in [7].

	Topics	Rew.	Searcher	Rer.	Rec.	MRR	nDCG@3	nDCG@10
1	Auto.	T5	BM25 _c	—	29.6	26.9	16.9	18.0
2			BM25 _c	BERT	29.6	43.8	31.3	29.5
3			DIR. _c	—	28.5	30.7	18.5	18.3
4			DIR. _c	BERT	28.5	40.4	29.8	27.1
5			BERT _c	—	40.4	34.2	23.6	23.5
6			SPLADE _c	—	46.7	45.6	35.1	32.7
Baseline Automatic at TREC CAsT 2020					27.6	40.8	30.0	27.7
Best Automatic Run at TREC CAsT 2020					—	59.3	45.8	—
7	Man.	—	BM25 _c	—	41.7	40.3	25.8	26.0
8			BM25 _c	BERT	41.7	58.4	43.7	40.7
9			DIR. _c	—	42.1	43.1	27.3	26.8
10			DIR. _c	BERT	42.1	59.4	42.8	40.3
11			BERT _c	—	56.4	50.8	35.6	34.7
12			SPLADE _c	—	61.5	62.4	47.8	44.9
Baseline Manual at TREC CAsT 2020					46.3	65.2	47.9	45.0
Best Manual Run at TREC CAsT 2020					—	68.4	53.0	—

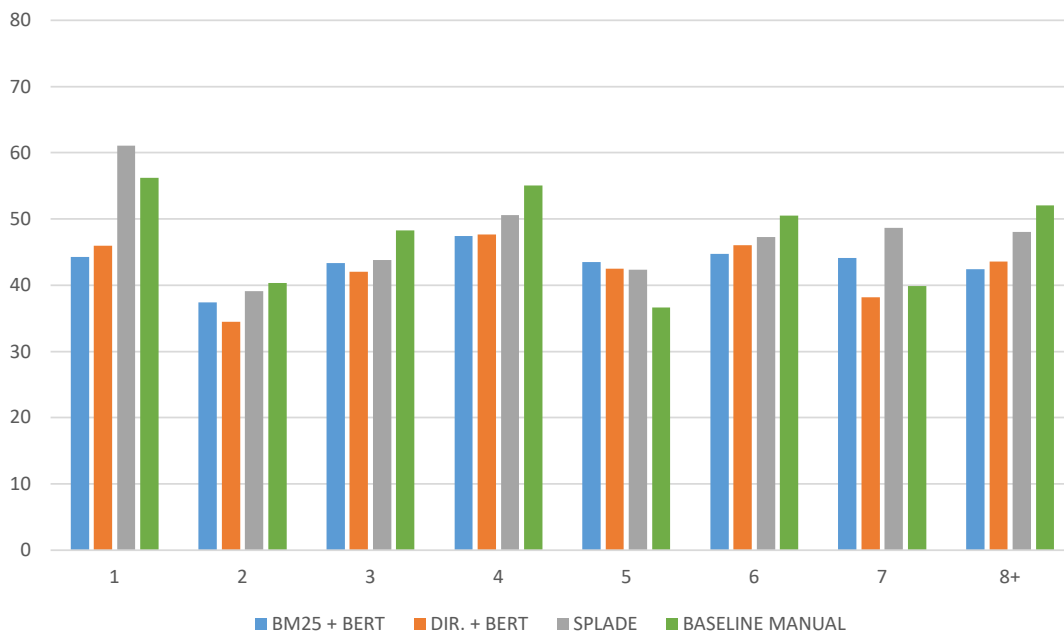
5.2. TREC CAST 2020 RESULTS

the MRR and nDCG@3 data for the best runs across the original submissions of TREC CAst 2020, as reported in the track overview [7].

The effectiveness of the retrieval systems as the conversation unfolds is evaluated in Figure 5.9a and 5.9b, respectively for automatic and manual runs. All systems perform well at the start of the conversation, whereas from turn 2 onward there is a significant reduction in the average nDCG@3 metric. It can be estimated as a 40 % - 50 % decrease for automatic runs, while it is smaller (17 % - 33 %) for manual. After the first turn, the performance remains roughly the same for the remaining of the conversation.



(a) Automatic Runs



(b) Manual Runs

Figure 5.9: Average nDCG@3 metric at varying conversation depths, on TREC CAst 2020 dataset. In the top plot, there are automatic runs, while the other displays manual runs. The considered runs are: BM25 with BERT (blue bar, on the left), Dirichlet LM with BERT (orange bar, in the center-left), SPLADE (grey bar, in the center-right), and the Baseline run across original submissions (green bar, on the right).



Conclusions

In this work, we have presented DECAF, a novel resource for conducting experiments within the CIS scenario. This work is motivated by the constantly growing plethora of heterogeneous CS systems that have been recently devised thanks to the advent of LLMs. DECAF has been designed to favour comparability between systems, fast prototyping and reproducibility, and in turn, alleviate the current reproducibility crisis. Therefore, DECAF has been designed around three key features: modularity, expandability and reproducibility. DECAF supports the fundamental building blocks that characterize modern CS systems and comes with a set of state-of-the-art components already implemented out of the box, including query rewriting, searching and re-ranking. The framework is also flexible enough to integrate additional components without much effort. We have evaluated several CS pipelines instantiated through DECAF on two well-known collections, TREC CAsT 2019 and 2020.

Future work will concern the extension of DECAF to support mixed-initiative tasks, such as those offered by TREC CAsT 2022.

References

- [1] Mohammad Aliannejadi et al. “Analysing Mixed Initiatives and Search Strategies during Conversational Search”. In: *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*. Ed. by Gianluca Demartini et al. ACM, 2021, pp. 16–26. DOI: 10 . 1145 / 3459637 . 3482231. URL: <https://doi.org/10.1145/3459637.3482231>.
- [2] Mohammad Aliannejadi et al. “Asking Clarifying Questions in Open-Domain Information-Seeking Conversations”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*. Ed. by Benjamin Piwowarski et al. ACM, 2019, pp. 475–484. DOI: 10 . 1145/3331184 . 3331265. URL: <https://doi.org/10.1145/3331184.3331265>.
- [3] Avishek Anand et al. “Conversational Search - A Report from Dagstuhl Seminar 19461”. In: *CoRR abs/2005.08658 (2020)*. arXiv: 2005 . 08658. URL: <https://arxiv.org/abs/2005.08658>.
- [4] Srinivas Bangalore, Giuseppe Di Fabbrizio, and Amanda Stent. “Learning the Structure of Task-Driven Human-Human Dialogs”. In: *IEEE Trans. Speech Audio Process.* 16.7 (2008), pp. 1249–1259. DOI: 10 . 1109/TASL . 2008 . 2001102. URL: <https://doi.org/10.1109/TASL.2008.2001102>.
- [5] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. “Reciprocal rank fusion outperforms condorcet and individual rank learning methods”. In: *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*. Ed. by James Allan et al. ACM, 2009, pp. 758–759. DOI: 10 . 1145/1571941 . 1572114. URL: <https://doi.org/10.1145/1571941.1572114>.

REFERENCES

- [6] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. “Are we really making much progress? A worrying analysis of recent neural recommendation approaches”. In: *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019*. Ed. by Toine Bogers et al. ACM, 2019, pp. 101–109. DOI: 10.1145/3298689.3347058. URL: <https://doi.org/10.1145/3298689.3347058>.
- [7] Jeffrey Dalton, Chenyan Xiong, and Jamie Callan. “CAst 2020: The Conversational Assistance Track Overview”. In: *Proceedings of the Twenty-Ninth Text REtrieval Conference, TREC 2020, Virtual Event [Gaithersburg, Maryland, USA], November 16-20, 2020*. Ed. by Ellen M. Voorhees and Angela Ellis. Vol. 1266. NIST Special Publication. National Institute of Standards and Technology (NIST), 2020. URL: <https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.C.pdf>.
- [8] Jeffrey Dalton, Chenyan Xiong, and Jamie Callan. “TREC CAst 2019: The Conversational Assistance Track Overview”. In: *CoRR abs/2003.13624 (2020)*. arXiv: 2003.13624. URL: <https://arxiv.org/abs/2003.13624>.
- [9] Jeffrey Dalton, Chenyan Xiong, and Jamie Callan. “TREC CAst 2021: The Conversational Assistance Track Overview”. In: *CoRR (2022)*, pp. 1–7. URL: https://www.cs.cmu.edu/~callan/Papers/trec22-Jeffrey_Dalton.pdf.
- [10] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *CoRR abs/1810.04805 (2018)*. arXiv: 1810.04805. URL: <http://arxiv.org/abs/1810.04805>.
- [11] Laura Dietz et al. “TREC Complex Answer Retrieval Overview”. In: *Proceedings of the Twenty-Seventh Text REtrieval Conference, TREC 2018, Gaithersburg, Maryland, USA, November 14-16, 2018*. Ed. by Ellen M. Voorhees and Angela Ellis. Vol. 500-331. NIST Special Publication. National Institute of Standards and Technology (NIST), 2018. URL: <https://trec.nist.gov/pubs/trec27/papers/Overview-CAR.pdf>.
- [12] Ahmed Elgohary, Denis Peskov, and Jordan L. Boyd-Graber. “Can You Unpack That? Learning to Rewrite Questions-in-Context”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by Kentaro Inui et

- al. Association for Computational Linguistics, 2019, pp. 5917–5923. DOI: 10.18653/v1/D19-1605. URL: <https://doi.org/10.18653/v1/D19-1605>.
- [13] Robert A. Fairthorne. “Automatic Retrieval of Recorded Information”. In: *Comput. J.* 1.1 (1958), pp. 36–41. DOI: 10.1093/comjnl/1.1.36. URL: <https://doi.org/10.1093/comjnl/1.1.36>.
- [14] Nicola Ferro. “Reproducibility Challenges in Information Retrieval Evaluation”. In: *ACM J. Data Inf. Qual.* 8.2 (2017), 8:1–8:4. DOI: 10.1145/3020206. URL: <https://doi.org/10.1145/3020206>.
- [15] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. “SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking”. In: *CoRR abs/2107.05720* (2021). arXiv: 2107.05720. URL: <https://arxiv.org/abs/2107.05720>.
- [16] Thibault Formal et al. “SPLADE v2: Sparse Lexical and Expansion Model for Information Retrieval”. In: *CoRR abs/2109.10086* (2021). arXiv: 2109.10086. URL: <https://arxiv.org/abs/2109.10086>.
- [17] Matt Gardner et al. “AllenNLP: A Deep Semantic Natural Language Processing Platform”. In: *CoRR abs/1803.07640* (2018). arXiv: 1803.07640. URL: <http://arxiv.org/abs/1803.07640>.
- [18] Jia-Chen Gu, Zhen-Hua Ling, and Quan Liu. “Utterance-to-Utterance Interactive Matching Network for Multi-Turn Response Selection in Retrieval-Based Chatbots”. In: *IEEE ACM Trans. Audio Speech Lang. Process.* 28 (2020), pp. 369–379. DOI: 10.1109/TASLP.2019.2955290. URL: <https://doi.org/10.1109/TASLP.2019.2955290>.
- [19] Kai Hui et al. “PACRR: A Position-Aware Neural IR Model for Relevance Matching”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Ed. by Martha Palmer, Rebecca Hwa, and Sebastian Riedel. Association for Computational Linguistics, 2017, pp. 1049–1058. DOI: 10.18653/v1/d17-1110. URL: <https://doi.org/10.18653/v1/d17-1110>.
- [20] Jeff Johnson, Matthijs Douze, and Hervé Jégou. “Billion-Scale Similarity Search with GPUs”. In: *IEEE Trans. Big Data* 7.3 (2021), pp. 535–547. DOI: 10.1109/TBDATA.2019.2921572. URL: <https://doi.org/10.1109/TBDATA.2019.2921572>.

REFERENCES

- [21] Sadegh Kharazmi et al. “Examining Additivity and Weak Baselines”. In: *ACM Trans. Inf. Syst.* 34.4 (2016), 23:1–23:18. DOI: 10 . 1145 / 2882782. URL: <https://doi.org/10.1145/2882782>.
- [22] Antonios Minas Krasakis et al. “Analysing the Effect of Clarifying Questions on Document Ranking in Conversational Search”. In: *ICTIR '20: The 2020 ACM SIGIR International Conference on the Theory of Information Retrieval, Virtual Event, Norway, September 14-17, 2020*. Ed. by Krisztian Balog et al. ACM, 2020, pp. 129–132. DOI: 10 . 1145 / 3409256 . 3409817. URL: <https://doi.org/10.1145/3409256.3409817>.
- [23] Juntao Li et al. “Dialogue History Matters! Personalized Response Selection in Multi-Turn Retrieval-Based Chatbots”. In: *ACM Trans. Inf. Syst.* 39.4 (2021), 45:1–45:25. DOI: 10 . 1145 / 3453183. URL: <https://doi.org/10.1145/3453183>.
- [24] Yongqi Li, Wenjie Li, and Liqiang Nie. “Dynamic Graph Reasoning for Conversational Open-Domain Question Answering”. In: *ACM Trans. Inf. Syst.* 40.4 (Jan. 2022). ISSN: 1046-8188. DOI: 10 . 1145 / 3498557. URL: <https://doi.org/10.1145/3498557>.
- [25] Jimmy Lin et al. “Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations”. In: *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*. Ed. by Fernando Diaz et al. ACM, 2021, pp. 2356–2362. DOI: 10 . 1145 / 3404835 . 3463238. URL: <https://doi.org/10.1145/3404835.3463238>.
- [26] Jimmy Lin et al. “Toward Reproducible Baselines: The Open-Source IR Reproducibility Challenge”. In: *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings*. Ed. by Nicola Ferro et al. Vol. 9626. Lecture Notes in Computer Science. Springer, 2016, pp. 408–420. DOI: 10 . 1007 / 978 - 3 - 319 - 30671 - 1_30. URL: https://doi.org/10.1007/978-3-319-30671-1%5C_30.
- [27] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. “Distilling Dense Representations for Ranking using Tightly-Coupled Teachers”. In: *CoRR* abs/2010.11386 (2020). arXiv: 2010.11386. URL: <https://arxiv.org/abs/2010.11386>.

- [28] Sheng-Chieh Lin et al. “Multi-Stage Conversational Passage Retrieval: An Approach to Fusing Term Importance Estimation and Neural Query Rewriting”. In: *ACM Trans. Inf. Syst.* 39.4 (2021), 48:1–48:29. DOI: 10.1145/3446426. URL: <https://doi.org/10.1145/3446426>.
- [29] Sean MacAvaney et al. “CEDR: Contextualized Embeddings for Document Ranking”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*. Ed. by Benjamin Piwowarski et al. ACM, 2019, pp. 1101–1104. DOI: 10.1145/3331184.3331317. URL: <https://doi.org/10.1145/3331184.3331317>.
- [30] Craig Macdonald and Nicola Tonellotto. “Declarative Experimentation in Information Retrieval using PyTerrier”. In: *ICTIR '20: The 2020 ACM SIGIR International Conference on the Theory of Information Retrieval, Virtual Event, Norway, September 14-17, 2020*. Ed. by Krisztian Balog et al. ACM, 2020, pp. 161–168. DOI: 10.1145/3409256.3409829. URL: <https://doi.org/10.1145/3409256.3409829>.
- [31] Ida Mele et al. “Adaptive utterance rewriting for conversational search”. In: *Inf. Process. Manag.* 58 (2021), p. 102682.
- [32] Ida Mele et al. “Topic Propagation in Conversational Search”. In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*. Ed. by Jimmy X. Huang et al. ACM, 2020, pp. 2057–2060. DOI: 10.1145/3397271.3401268. URL: <https://doi.org/10.1145/3397271.3401268>.
- [33] Tri Nguyen et al. “MS MARCO: A Human Generated Machine Reading Comprehension Dataset”. In: *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016*. Ed. by Tarek Richard Besold et al. Vol. 1773. CEUR Workshop Proceedings. CEUR-WS.org, 2016. URL: http://ceur-ws.org/Vol-1773/CoCoNIPS%5C_2016%5C_paper9.pdf.
- [34] Rodrigo Frassetto Nogueira and Kyunghyun Cho. “Passage Re-ranking with BERT”. In: *CoRR abs/1901.04085* (2019). arXiv: 1901.04085. URL: <http://arxiv.org/abs/1901.04085>.

REFERENCES

- [35] Shon Otmazgin, Arie Cattan, and Yoav Goldberg. “F-COREF: Fast, Accurate and Easy to Use Coreference Resolution”. In: *CoRR abs/2209.04280* (2022). DOI: 10.48550/arXiv.2209.04280. arXiv: 2209.04280. URL: <https://doi.org/10.48550/arXiv.2209.04280>.
- [36] Shon Otmazgin, Arie Cattan, and Yoav Goldberg. “LingMess: Linguistically Informed Multi Expert Scorers for Coreference Resolution”. In: *CoRR abs/2205.12644* (2022). DOI: 10.48550/arXiv.2205.12644. arXiv: 2205.12644. URL: <https://doi.org/10.48550/arXiv.2205.12644>.
- [37] Iadh Ounis et al. “Terrier Information Retrieval Platform”. In: *Advances in Information Retrieval, 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005, Proceedings*. Ed. by David E. Losada and Juan M. Fernández-Luna. Vol. 3408. Lecture Notes in Computer Science. Springer, 2005, pp. 517–519. DOI: 10.1007/978-3-540-31865-1_37. URL: https://doi.org/10.1007/978-3-540-31865-1_37.
- [38] Gustavo Penha and Claudia Hauff. “Challenges in the Evaluation of Conversational Search Systems”. In: *Proceedings of the KDD 2020 Workshop on Conversational Systems Towards Mainstream Adoption co-located with the 26TH ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD 2020), Virtual Workshop, August 24, 2020*. Ed. by Giuseppe Di Fabbrizio et al. Vol. 2666. CEUR Workshop Proceedings. CEUR-WS.org, 2020. URL: http://ceur-ws.org/Vol-2666/KDD%5C_Converse20%5C_paper%5C_5.pdf.
- [39] Fabio Petroni et al. “KILT: a Benchmark for Knowledge Intensive Language Tasks”. In: *CoRR abs/2009.02252* (2020). arXiv: 2009.02252. URL: <https://arxiv.org/abs/2009.02252>.
- [40] Alec Radford et al. “Improving Language Understanding by Generative Pre-Training”. In: 2018.
- [41] Filip Radlinski and Nick Craswell. “A Theoretical Framework for Conversational Search”. In: *Proc. CHIIR*. New York, NY, USA: ACM, 2017, pp. 117–126.
- [42] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *CoRR abs/1910.10683* (2019). arXiv: 1910.10683. URL: <http://arxiv.org/abs/1910.10683>.

- [43] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. URL: <http://jmlr.org/papers/v21/20-074.html>.
- [44] Gonçalo Raposo et al. “Question Rewriting? Assessing Its Importance for Conversational Question Answering”. In: *Advances in Information Retrieval - 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10-14, 2022, Proceedings, Part II*. Ed. by Matthias Hagen et al. Vol. 13186. Lecture Notes in Computer Science. Springer, 2022, pp. 199–206. DOI: 10.1007/978-3-030-99739-7_23. URL: https://doi.org/10.1007/978-3-030-99739-7%5C_23.
- [45] Stephen E. Robertson and Hugo Zaragoza. “The Probabilistic Relevance Framework: BM25 and Beyond”. In: *Found. Trends Inf. Retr.* 3.4 (2009), pp. 333–389. DOI: 10.1561/1500000019. URL: <https://doi.org/10.1561/1500000019>.
- [46] Gerard Salton and Chris Buckley. “Term-Weighting Approaches in Automatic Text Retrieval”. In: *Inf. Process. Manag.* 24.5 (1988), pp. 513–523.
- [47] Gerard. Salton. *Automatic Information Organization and Retrieval*. McGraw Hill Text, 1968. ISBN: 0070544859.
- [48] Chongyang Tao et al. “Multi-Representation Fusion Network for Multi-Turn Response Selection in Retrieval-Based Chatbots”. In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*. Ed. by J. Shane Culpepper et al. ACM, 2019, pp. 267–275. DOI: 10.1145/3289600.3290985. URL: <https://doi.org/10.1145/3289600.3290985>.
- [49] Svitlana Vakulenko et al. “Question Rewriting for Conversational Question Answering”. In: *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021*. Ed. by Liane Lewin-Eytan et al. ACM, 2021, pp. 355–363. DOI: 10.1145/3437963.3441748. URL: <https://doi.org/10.1145/3437963.3441748>.
- [50] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 5998–6008. URL: <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>.

REFERENCES

- [51] Nikos Voskarides et al. "Query Resolution for Conversational Search with Limited Supervision". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 921–930. ISBN: 9781450380164. URL: <https://doi.org/10.1145/3397271.3401130>.
- [52] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*. Ed. by Qun Liu and David Schlangen. Association for Computational Linguistics, 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: <https://doi.org/10.18653/v1/2020.emnlp-demos.6>.
- [53] Yu Wu et al. "Sequential Matching Network: A New Architecture for Multi-turn Response Selection in Retrieval-Based Chatbots". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*. Ed. by Regina Barzilay and Min-Yen Kan. Association for Computational Linguistics, 2017, pp. 496–505. DOI: 10.18653/v1/P17-1046. URL: <https://doi.org/10.18653/v1/P17-1046>.
- [54] Chenyan Xiong et al. "End-to-End Neural Ad-hoc Ranking with Kernel Pooling". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*. Ed. by Noriko Kando et al. ACM, 2017, pp. 55–64. DOI: 10.1145/3077136.3080809. URL: <https://doi.org/10.1145/3077136.3080809>.
- [55] Lee Xiong et al. "Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: <https://openreview.net/forum?id=zeFrfgYzln>.
- [56] Rui Yan. "'Chitty-Chitty-Chat Bot': Deep Learning for Conversational AI". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Ed. by Jérôme Lang. ijcai.org, 2018, pp. 5520–5526. DOI: 10.24963/ijcai.2018/778. URL: <https://doi.org/10.24963/ijcai.2018/778>.

- [57] Jheng-Hong Yang et al. “Query and Answer Expansion from Conversation History”. In: *TREC*. 2019.
- [58] Peilin Yang, Hui Fang, and Jimmy Lin. “Anserini: Enabling the Use of Lucene for Information Retrieval Research”. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*. Ed. by Noriko Kando et al. ACM, 2017, pp. 1253–1256. DOI: 10.1145/3077136.3080721. URL: <https://doi.org/10.1145/3077136.3080721>.
- [59] Peilin Yang, Hui Fang, and Jimmy Lin. “Anserini: Reproducible Ranking Baselines Using Lucene”. In: *ACM J. Data Inf. Qual.* 10.4 (2018), 16:1–16:20. DOI: 10.1145/3239571. URL: <https://doi.org/10.1145/3239571>.
- [60] Wei Yang et al. “Critically Examining the “Neural Hype”: Weak Baselines and the Additivity of Effectiveness Gains from Neural Ranking Models”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*. Ed. by Benjamin Piwowarski et al. ACM, 2019, pp. 1129–1132. DOI: 10.1145/3331184.3331340. URL: <https://doi.org/10.1145/3331184.3331340>.
- [61] Zhou Yang et al. “A Deep Top-K Relevance Matching Model for Ad-hoc Retrieval”. In: *Information Retrieval - 24th China Conference, CCIR 2018, Guilin, China, September 27-29, 2018, Proceedings*. Ed. by Shichao Zhang et al. Vol. 11168. Lecture Notes in Computer Science. Springer, 2018, pp. 16–27. DOI: 10.1007/978-3-030-01012-6_2. URL: https://doi.org/10.1007/978-3-030-01012-6_2.
- [62] Shi Yu et al. “Few-Shot Conversational Dense Retrieval”. In: *SIGIR ’21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*. Ed. by Fernando Diaz et al. ACM, 2021, pp. 829–838. DOI: 10.1145/3404835.3462856. URL: <https://doi.org/10.1145/3404835.3462856>.
- [63] Zhou Yu et al. “Strategy and Policy Learning for Non-Task-Oriented Conversational Systems”. In: *Proceedings of the SIGDIAL 2016 Conference, The 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 13-15 September 2016, Los Angeles, CA, USA*. The Association for Computer

REFERENCES

- Linguistics, 2016, pp. 404–412. DOI: 10.18653/v1/w16-3649. URL: <https://doi.org/10.18653/v1/w16-3649>.
- [64] Hamed Zamani et al. “Conversational Information Seeking. An Introduction to Conversational Search, Recommendation, and Question Answering”. In: *arXiv.org, Information Retrieval (cs.IR)* arXiv:2201.08808 (Jan. 2022).
- [65] Cheng Xiang Zhai. “Statistical Language Models for Information Retrieval: A Critical Review”. In: *Found. Trends Inf. Retr.* 2.3 (2008), pp. 137–213. DOI: 10.1561/15000000008. URL: <https://doi.org/10.1561/15000000008>.
- [66] Jingtao Zhan et al. “Optimizing Dense Retrieval Model Training with Hard Negatives”. In: *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*. Ed. by Fernando Diaz et al. ACM, 2021, pp. 1503–1512. DOI: 10.1145/3404835.3462880. URL: <https://doi.org/10.1145/3404835.3462880>.

Acknowledgments