DECAF: a Modular and Extensible Conversational Search Framework

Marco Alessio
marco.alessio.1@studenti.unipd.it
University of Padova
Padova, Italy

Guglielmo Faggioli
faggioli@dei.unipd.it
University of Padova
Padova, Italy

Nicola Ferro
ferro@dei.unipd.it
University of Padova
Padova, Italy

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 conversational search systems with custom implementations and variants of information retrieval models. This exacerbates the reproducibility crisis already observed in several research areas, including Information Retrieval (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 conversational search 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 BoW and dense paradigms, and re-ranking functions. Our framework is tested on two well-known conversational collections: TREC CAsT 2019 and TREC CAsT 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 conversational search task and the definition of a modular framework for its implementation.

CCS CONCEPTS

• **Information systems → Search engine architectures and scalability**.

ACM Reference Format:
Marco Alessio, Guglielmo Faggioli, and Nicola Ferro. 2023. DECAF: a Modular and Extensible Conversational Search Framework. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3539618.3591913

SIGIR '23, July 23–27, 2023, Taipei, Taiwan

ACM ISBN 978-1-4503-9408-6/23/07…$15.00
https://doi.org/10.1145/3539618.3591913

1 INTRODUCTION

Conversational Search (CS) [3, 56] is an emerging paradigm which is drastically innovating Information Retrieval (IR) by allowing users to issue their queries to the system in the form of 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 sentences. Nevertheless, it also presents important challenges due to the presence of complex speech structures in the utterances, such as anaphoras, ellipses or coreferences. Therefore, the system needs to address these natural language phenomena by keeping track of the conversation state.

The advent of Neural Information Retrieval (NIR) models, fueled by Large Language Models (LLMs), has been helping to overcome some of these challenges and to foster a capillary adoption of CS in many different scenarios. Such techniques, given their complexity, were originally used mostly for re-ranking [17, 46, 53]. With their improvement both in terms of effectiveness and efficiency, we also witnessed their adoption as first-stage retrieval systems [14, 47, 58]. The plethora of CS systems that rely on these building blocks allow for addressing the user's information need in a number of different scenarios. Nevertheless, 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, implemented using a wide range of different languages and frameworks, often difficult to integrate together, if not impossible at all. This causes a number of challenges both in the development of CS systems and in their experimentation. Firstly, it is difficult to combine various state-of-the-art components together, since they often come from different and/or incompatible libraries and packages; in turn, this hampers the development of new and competitive approaches. Secondly, alternative implementations of a component often lead to different performance or behavior; in turn, this hampers the comparability of different end-to-end solutions, integrating alternative versions of the same components. Thirdly, it is extremely difficult, if not impossible, to conduct systematic experiments where different components are combined in all the possible ways, in a grid-like way [11], in order to break down the contribution of different components and analyse their interactions [12]. Finally, it hinders the reproducibility of experiments, exacerbating the already prevalent reproducibility crisis in current research [5, 10, 19, 52].

To address these issues, we propose **DECAF** – *mo**D**ular and **E**xtensible **C**onversational sea**A**rch **F**ramework*, 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, search – both under the Bag-of-Words (BoW) and dense paradigms – and re-ranking. While written primarily in Java to ensure compatibility with traditional IR libraries, such as Lucene[1], Terrier [34], and Anserini [23], it also supports modern Artificial Intelligence (AI), Machine Learning (ML), and LLM-based techniques thanks to the seamless integration of Python scripts. The framework already includes multiple state-of-the-art (sota) 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 TREC CAsT 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 TREC CAsT 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.
- 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.

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

2 RELATED WORKS

We report in the remainder of this section the main endeavours and related work concerning three topics: the main efforts in CS, the principal evaluation campaigns in this setting – used as a reference point to design DECAF, and the related IR frameworks.

2.1 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 [36] as the dialogue unfolds. In fact, 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, 30].

Depending on the task they absolve, conversational agents are commonly divided into chit-chat bots [48, 55] and task-oriented systems [4, 16, 35]. 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 [16, 41, 45] and systems that construct the response by combining multiple retrieved sources using summarization approaches such as T5 [37].

The multi-turn conversational task is characterized by the importance given to the "context" [21, 29, 38, 42]. The context 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 [22, 26, 30, 43, 49]. Another approach to modelling the context consists of adopting approaches based on dense retrieval models. For example, Yu et al. [54] employ the teacher-student framework to learn a student query encoder to use in conjunction with a standard ad hoc dense retrieval teacher model, such as ANCE [47], TCT-ColBERT [25] 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 [54].

2.2 TREC Conversational Assistance Track

The Conversational Assistance Track (CAsT) at the Text Retrieval Conference (TREC) was held for the first time in 2019 [7] 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 [6].

The main task evaluated in CAsT is passages (in 2019 [7] and 2020 [6]) or documents (since 2021 [8]) retrieval from a corpus composed of multiple sub-corpora, such as MS-MARCO, TREC CAR, Wikipedia, 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 [7], while, from the second edition onward [6, 8] conversations take into consideration the responses given by the system as well. The 2022 edition saw the introduction of a mixed-initiative [1, 20] 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.

2.3 IR Frameworks

Several IR libraries, such as Lucene, Terrier [34], and Anserini [23, 50, 51] 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

[1] https://lucene.apache.org/

communities underneath. With the advent of ML solutions and the increased popularity of Python, new wrappers around traditional frameworks were designed, such as Pyserini [24] and PyTerrier [28]. Pyserini [24] 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 [28] 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. Moreover, Chatty Goose [59] is a Python framework for executing conversational search experiments, providing reproducible pipelines that practitioners can build on top of. Despite being modular, Chatty Goose is bound to use Pyserini and Pygaggle for performing the first-stage retrieval and the reranking phases, respectively. For this reason, Chatty Goose is not a general and solution-agnostic CS framework.

We are interested in developing a solution specifically designed for CS but capable of exploiting the advantages of the already existing frameworks. Therefore, 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 we can 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 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 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 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 performing pronoun disambiguation and 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 and re-ranking 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

Requirements. The core of DECAF has been developed in Java, with the integration of Python for machine-learning-oriented functionalities. It requires Java Development Kit (JDK) 11 and Python 3.8 for execution. The framework is built upon Lucene 8.8.1[2].

Every component is expected to implement the Java interface of the specific module, which defines one or more methods specific to the performed job. The components implemented with DECAF and described in the remainder of this section can be used by practitioners both as baselines as well as templates to guide practitioners in extending DECAF. Every component of the framework has 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 Section 4).

3.2.1 Index Pipeline.

Corpus Parser. We implement three components that perform corpus parsing. The first processes the passages contained within MS-MARCO version 1[3] dataset. DECAF also supports duplicate removal: the discarded documents are specified through an input file. Another parser addresses the paragraph corpus of TREC CAR v2.0 [4]. The third parser processes any corpus based on tab-separated files using the "`ID <tab> Text <newline>`" format.

Furthermore, it might be necessary to index documents from multiple sources with different formats at once. The last corpus 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 a BoW indexer, a SPLADE indexer, and a 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

[2] https://lucene.apache.org/core/8_8_1/index.html
[3] https://msmarco.blob.core.windows.net/msmarcoranking/collection.tar.gz
[4] https://trec-car.cs.unh.edu/datareleases/v2.0/paragraphCorpus.v2.0.tar.xz

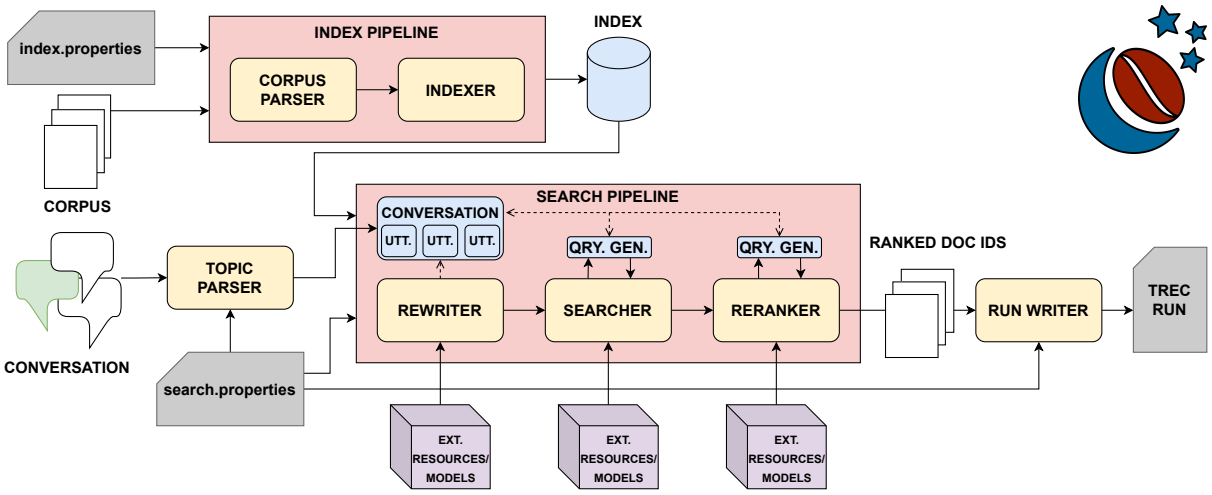


Figure 1: Architecture of DECAF.

model[5]. It replaces the standard tokenization and analysis pipeline performed by Lucene with the SPLADE model inference. We need a separate interface to operate with SPLADE 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 [44] and Facebook AI Similarity Search (FAISS) [18]. The former is an open-source library providing APIs and tools to download, train and use sota machine-learning 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.

3.2.2 *Search Pipeline.*

Conversation and Utterance Components. To provide a unified interface to the data, we define two 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. The latter provides utilities to access the data and groups together 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 implemented components 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 we have to operate (e.g., we want to rewrite or we are retrieving the documents for). This approach ensures great flexibility in the design of the components 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 the framework, since each component can behave as a black-box building block operating only on the Conversation and Utterance objects.

Topics Parser. DECAF provides five topic parsers designed explicitly to handle TREC CAsT 2019 and TREC CAsT 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 preprocessing 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 Utterances that compose it.

Rewriter. In DECAF, we provide two sota rewriting approaches using off-the-shelf resources, either employing coreference resolution libraries or pre-trained T5 models.

In implementation terms, there are two library-based components that carry out coreference resolution, which differ solely for the library used to perform the operation. In particular, we implement one component based on AllenNLP framework [15] and one using Fastcoref. Fastcoref is a coreference resolution utility based on the LingMess [33] architecture, providing state-of-the-art coreference accuracy [32]. In Table 1, we dub this approach `CR`.

[5]https://huggingface.co/naver/efficient-splade-V-large-doc

For the second approach, we employ a T5 model, 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 natural language processing tasks, including anaphora and coreference resolution [37]. The particular instance of T5 used in the experimental part is publicly available and pre-trained specifically for conversational search question rewriting[6].

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. 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 [29]. A second query generator provided within DECAF considers the concatenation of the rewritten text for all previous utterances of the current conversation. The final query generator implemented (dubbed `Current` in Table 2) considers only the – possibly rewritten – text of the current utterance, without taking into account any of the previous ones. To operate, the query generator access the Utterance object referring to the current utterance, and possibly to the Utterance objects defined for 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 [39], Language Model (LM) [57] or the Vector Space Model [40]. For example, in the experimental benchmarking reported in Section 5, we exploit BM25. BM25 [39] is 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 Section 4). The second similarity function that we take into consideration is SPLADE. SPLADE [13, 14] is a sparse NIR model that learns sparse representation for both queries and documents via the BERT MLM head and sparse regularization. It is particularly appealing for the first stage retrieval phase, thanks to the simplicity, efficiency, and explainability of sparse representations. We separate it from the previous component, even though they are both BoW because SPLADE requires computing the BoW sparse representation of the query. In our experimental analysis, we use a publicly available set of weights[7]. Finally, we implement a component for dense retrieval that can be used with FAISS indexes. In particular, we instantiate it 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 [9]. Notice that, we exploit a publicly available BERT bi-encoder instance fine-tuned specifically for IR[8]. 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) [44] library to apply machine-learning 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. Notice that, as it will be detailed later in Section 4, the specific transformers model used can be chosen at runtime, by specifying it in the properties file. We experiment with BERT, since several works already observed the effectiveness of BERT for the re-ranking task in the CS and IR domains [27, 29, 31].

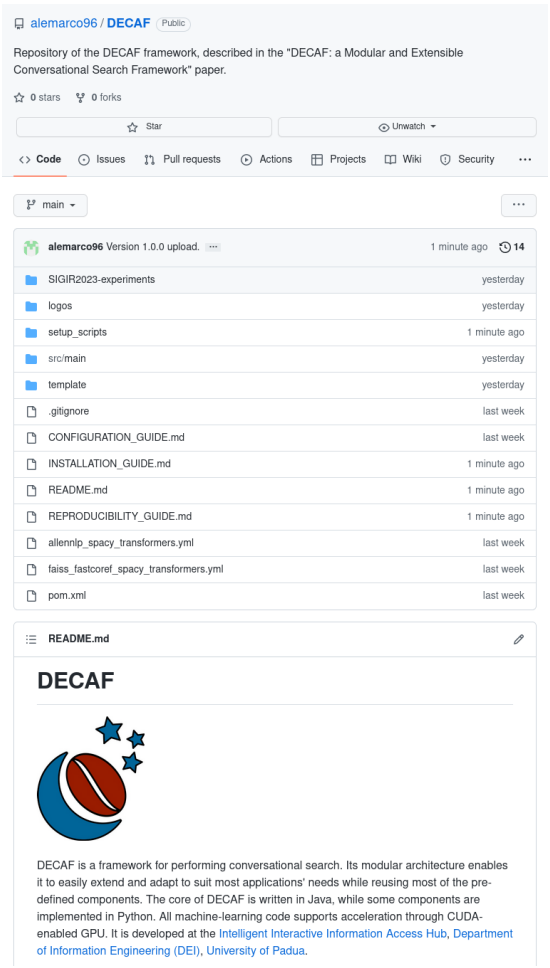
At the run-time, the Transformers component can be customized depending on the practitioner's needs. In particular, it is possible to set different similarity functions enabling to adapt to different transformer models. The similarity functions currently implemented are cosine similarity, dot product and Euclidean distance. This component can optionally perform run fusion between the newly-generated ranked list with the one given by the Searcher. There is an option to disable run fusion whether it is not needed. 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. 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

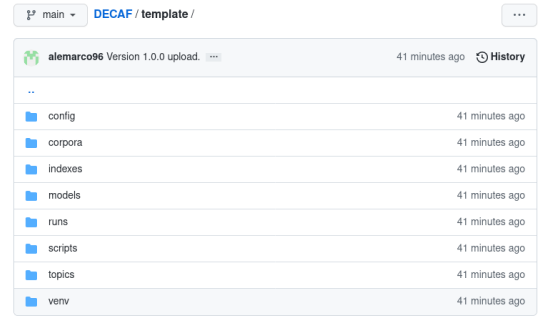
[6]https://huggingface.co/castorini/t5-base-canard

[7]https://huggingface.co/naver/efficient-splade-V-large-query

[8]https://huggingface.co/sebastian-hofstaetter/distilbert-dot-tas_b-b256-msmarco



(a) Main directory of the repository.



(b) *template* sub-directory.

Figure 2: Screenshots of the DECAF repository.

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 Git Repository

DECAF is available as open source under the *Creative Commons Attribution-ShareAlike 4.0 International License*[9] at the following address: https://github.com/alemarco96/DECAF.

As shown in Figure 2, 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 the configuration files for conducting experiments. Regarding the reproducibility of the experiments shown in Section 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. We describe here such directories – to ease the framework instantiation, they can be found on the *template* sub-directory of DECAF repository[10].

- 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 th 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 *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

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

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

[9] http://creativecommons.org/licenses/by-sa/4.0/

[10] In case of highly customized scenarios, the paths to these directories, as well as specific files, could also be manually set through the configuration files.

Configuration 1: index.properties

```
launch.corpus = CAsT1920

# Path to the corpus files
launch.corpus.CAsT1920.msmarco_corpus_filename =
    /path/to/location
launch.corpus.CAsT1920.msmarco_duplicate_filename =
    /path/to/location
launch.corpus.CAsT1920.treccar_corpus_filename =
    /path/to/location

launch.indexer = BoWIndexer

# Path to the directory containing the index
launch.indexer.BoWIndexer.index_directory =
    /path/to/location

# Text normalization component
launch.indexer.BoWIndexer.analyzer = English

# Similarity function used and other parameters
launch.indexer.BoWIndexer.similarity = BM25
launch.indexer.BoWIndexer.similarity.BM25.k1 = 0.82
launch.indexer.BoWIndexer.similarity.BM25.b = 0.4
launch.indexer.BoWIndexer.chunks_size = 1000000

launch.num_threads = 8
```

Figure 3: An example of configuration file snippet that allows for configuring the indexing phase.

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.

4.1 Index Phase

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 3 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 Section 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 Search Phase

Configuration 2: search.properties

```
launch.topics = AutCAsT19

launch.rewriter = T5
launch.rewriter.T5.model = t5-base-canard
launch.rewriter.T5.max_tokens = 512

launch.searcher = BoWSearcher

# Path to the directory containing the index
launch.searcher.BoWSearcher.index_directory =
    /path/to/location

# Text normalization component
launch.searcher.BoWSearcher.analyzer = English

# Similarity function used and its parameters
launch.searcher.BoWSearcher.similarity = BM25
launch.searcher.BoWSearcher.similarity.BM25.k1 = 0.82
launch.searcher.BoWSearcher.similarity.BM25.b = 0.4

# The query generator component
launch.searcher.BoWSearcher.query = Current

launch.reranker = Transformers

# The model to use with its parameters
launch.reranker.Transformers.model =
    distilbert-dot-tas_b-b256-msmarco
launch.reranker.Transformers.vector_size = 768
launch.reranker.Transformers.max_tokens = 512
launch.reranker.Transformers.similarity = dot

# The query generator and fusion components
launch.reranker.Transformers.query = Current
launch.reranker.Transformers.fusion = No

launch.run_writer = TrecEval
launch.run_writer.TrecEval.run_id = id_of_the_run

launch.num_documents = 100
launch.num_threads = 8
```

Figure 4: An example of a configuration file snippet that allows for configuring the search phase.

In this section, we describe how to customize the search phase operations using the `search.properties` configuration file. Figure 4 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

Table 1: Rewriters and corresponding configuration parameters, used in the evaluation of DECAF.

ID	Rewriter conf.	Description
—	rewriter: No	No rewriting is applied at all.
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.

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.

5 EXPERIMENTAL RESULTS

The framework has been tested on TREC CAsT 2019 and TREC CAsT 2020 settings, two popular benchmarks for evaluating conversation information-seeking systems. We experiment with multiple combinations of components and parameters, summarized in Tables 1, 2, and 3. In our experiments, we retrieved and reranked the top 100 documents for each query. Following the procedure used for CAsT evaluation campaign [6, 7], we report four widely used measures: Recall (R)@100, Mean Reciprocal Rank (MRR), normalized Discounted Cumulative Gain (nDCG)@3, and nDCG@10, computed using the `trec_eval`[11] tool. Following the official evaluation settings on TREC CAsT 2020 dataset, we consider documents as relevant if their relevance score is ≥ 2 [6].

5.1 TREC CAsT 2019 Results

Table 4 shows the experimental results on TREC CAsT 2019. The initial experiment (#1) evaluates the performance of the BM25 model on automatic utterances: the system performs poorly due to the lack of contextual information. Results are in line with those observed by [7] with respect to the baseline used for TREC CAsT 2019 (LMs). Experiments #2 and #3 add the Rewriter component to the pipeline, which brings forward implied substantives into the text of the utterance, resulting in improved performance. Run #3 employs T5 and achieves double the performance w.r.t the automatic baseline across all measures. The FLC variant (used in Experiments #4 and #5) combines the first, previous and current utterances' text. This simple heuristic is beneficial, particularly in cases where the rewriter fails to correctly expand and disambiguate pronouns with the relative entity. Again, results are consistent with the observations made in the TREC CAsT 2019 overview [7]. Experiments #6 to #9 demonstrate that performing an additional re-ranking step using a BERT model can further improve performance for all experimental settings. A comparison between experiment #1 and #6 shows that, despite the absence of any form of rewriting, the performance is significantly improved by using BERT as reranker, thus

[11] https://github.com/usnistgov/trec_eval

Table 2: Ranking functions and corresponding configuration parameters, used in the evaluation of DECAF.

ID	Ranker 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.
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 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 product 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.

alleviating the issues related to the "context representation" introduced by the CS setting. The performance of our implementation of the BERT-based re-ranking strategy is similar to the average performance of systems submitted at TREC CAsT 2019 [7]. If the same BERT model is used for first-stage dense retrieval (Experiment #10), it achieves performance similar to a standard lexical model such as BM25 (experiment #3). Experiments #11 and #12 evaluate SPLADE as a first-stage retrieval method. The particular SPLADE instance employed 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

Table 4: List of all experiments conducted 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.	—	BM25 C	—	19.9	32.0	14.2	14.4
2		CR	BM25 C	—	35.3	51.5	25.9	25.4
3		T5	BM25 C	—	42.8	64.0	33.9	32.1
4		—	BM25 FLC	—	23.9	41.0	19.1	18.4
5		T5	BM25 FLC	—	45.0	65.6	34.4	32.5
6		—	BM25 C	BERT	19.9	48.0	28.4	24.9
7		T5	BM25 C	BERT	42.8	79.3	50.4	45.2
8		—	BM25 FLC	BERT	23.9	51.7	30.9	27.5
9		T5	BM25 FLC	BERT	45.0	79.4	50.3	46.0
10		T5	BERT C	—	43.2	52.3	30.4	33.1
11		—	SPLADE C	—	24.8	44.8	27.5	26.7
12		T5	SPLADE C	—	**51.5**	**79.9**	**52.3**	**50.1**
Baseline Automatic at TREC CAsT 2019					21.4	31.8	14.8	15.9
Best Automatic Run at TREC CAsT 2019					41.2	71.1	42.4	40.6
13	Man.	—	BM25 C	—	47.8	66.7	35.4	34.5
14		—	BM25 C	BERT	47.8	82.5	54.4	48.2
15		—	BERT C	—	46.4	54.3	32.8	35.5
16		—	SPLADE C	—	54.9	84.3	56.6	53.5
Best Manual Run at TREC CAsT 2019					**56.7**	**88.4**	**59.3**	**60.6**

Table 5: 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.

	Topics	Rew.	Searcher	Rer.	Rec.	MRR	nDCG@3	nDCG@10
1	Auto.	T5	BM25 C	—	29.6	26.9	16.9	18.0
2		T5	BM25 C	BERT	29.6	43.8	31.3	29.5
3		T5	BERT C	—	40.4	34.2	23.6	23.5
4		T5	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					**54.3**	**59.3**	**45.8**	**44.6**
5	Man.	—	BM25 C	—	41.7	40.3	25.8	26.0
6		—	BM25 C	BERT	41.7	58.4	43.7	40.7
7		—	BERT C	—	56.4	50.8	35.6	34.7
8		—	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					58.1	**68.4**	**53.0**	**47.7**

queries. Test #12 shows that SPLADE achieves the best results, outperforming BM25 with BERT (experiment #9) by 14.4% in terms of recall, 4.0% for nDCG@3, and 8.9% for nDCG@10 when the T5-based rewriter is used. It also surpasses all original TREC CAsT 2019 automatic submissions, obtaining an improvement against the Best Automatic (BA) of respectively 25.0%, 12.4%, 23.3%, and 23.4% for the four measures considered.

We now focus on the second part of Table 4, 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. In fact, 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 8.6% for recall, 4.5% for MRR, 8.0% for nDCG@3 and 6.9% for nDCG@10. These experiments show 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, 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, especially for the nDCG@10 measure with a difference of 13.3%.

5.2 TREC CAsT 2020 Results

Table 5 shows the experimental results on TREC CAsT 2020. Due to space reasons, 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. Once again, the dense retrieval approach gives the lowest overall scores, followed by BM25 with BERT re-ranking, while SPLADE produces the best results. As observed for TREC CAsT 2019, BM25 is the lowest performing method, followed by BERT in first-stage retrieval. When it comes to using BERT as reranker, on the other hand, we observe a sensible improvement compared to both BM25 (+69.4% in nDCG@3) and BERT in the first stage retrieval (+22.8% in nDCG@3). 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. 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 the data for the best runs across the original submissions of TREC CAsT 2020, as reported in the track overview [6].

6 CONCLUSIONS

In this work, we have presented DECAF, a novel resource for conducting experiments within the Conversational Information Seeking (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 TREC CAsT 2020 .

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

ACKNOWLEDGMENTS

The work was partially supported by University of Padova Strategic Research Infrastructure Grant 2017: "CAPRI: Calcolo ad Alte Prestazioni per la Ricerca e l'Innovazione"

REFERENCES

[1] Mohammad Aliannejadi, Leif Azzopardi, Hamed Zamani, Evangelos Kanoulas, Paul Thomas, and Nick Craswell. 2021. 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*, Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong (Eds.). ACM, 16–26. https://doi.org/10.1145/3459637.3482231

[2] Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W. Bruce Croft. 2019. 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*, Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer (Eds.). ACM, 475–484. https://doi.org/10.1145/3331184.3331265

[3] Avishek Anand, Lawrence Cavedon, Matthias Hagen, Hideo Joho, Mark Sanderson, and Benno Stein. 2020. Conversational Search - A Report from Dagstuhl Seminar 19461. *CoRR* abs/2005.08658 (2020). arXiv:2005.08658 https://arxiv.org/abs/2005.08658

[4] Srinivas Bangalore, Giuseppe Di Fabbrizio, and Amanda Stent. 2008. Learning the Structure of Task-Driven Human-Human Dialogs. *IEEE Trans. Speech Audio Process.* 16, 7 (2008), 1249–1259. https://doi.org/10.1109/TASL.2008.2001102

[5] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. 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*, Toine Bogers, Alan Said, Peter Brusilovsky, and Domonkos Tikk (Eds.). ACM, 101–109. https://doi.org/10.1145/3298689.3347058

[6] Jeffrey Dalton, Chenyan Xiong, and Jamie Callan. 2020. 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 (NIST Special Publication, Vol. 1266)*, Ellen M. Voorhees and Angela Ellis (Eds.). National Institute of Standards and Technology (NIST). https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.C.pdf

[7] Jeffrey Dalton, Chenyan Xiong, and Jamie Callan. 2020. TREC CAsT 2019: The Conversational Assistance Track Overview. *CoRR* abs/2003.13624 (2020). arXiv:2003.13624 https://arxiv.org/abs/2003.13624

[8] Jeffrey Dalton, Chenyan Xiong, and Jamie Callan. 2022. TREC CAsT 2021: The Conversational Assistance Track Overview. *CoRR* (2022), 1–7. https://www.cs.cmu.edu/~callan/Papers/trec22-Jeffrey_Dalton.pdf

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[10] Nicola Ferro. 2017. Reproducibility Challenges in Information Retrieval Evaluation. *ACM J. Data Inf. Qual.* 8, 2 (2017), 8:1–8:4. https://doi.org/10.1145/3020206

[11] Nicola Ferro and Donna Harman. 2009. CLEF 2009: Grid@CLEF Pilot Track Overview. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments, 10th Workshop of the Cross-Language Evaluation Forum, CLEF 2009, Corfu, Greece, September 30 - October 2, 2009, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 6241)*, Carol Peters, Giorgio Maria Di Nunzio, Mikko Kurimo, Thomas Mandl, Djamel Mostefa, Anselmo Peñas, and Giovanna Roda (Eds.). Springer, 552–565. https://doi.org/10.1007/978-3-642-15754-7_68

[12] Nicola Ferro and Gianmaria Silvello. 2016. A General Linear Mixed Models Approach to Study System Component Effects. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, Raffaele Perego, Fabrizio Sebastiani, Javed A. Aslam, Ian Ruthven, and Justin Zobel (Eds.). ACM, 25–34. https://doi.org/10.1145/2911451.2911530

[13] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE v2: Sparse Lexical and Expansion Model for Information Retrieval. *CoRR* abs/2109.10086 (2021). arXiv:2109.10086 https://arxiv.org/abs/2109.10086

[14] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. *CoRR* abs/2107.05720 (2021). arXiv:2107.05720 https://arxiv.org/abs/2107.05720

[15] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A Deep Semantic Natural Language Processing Platform. *CoRR* abs/1803.07640 (2018). arXiv:1803.07640 http://arxiv.org/abs/1803.07640

[16] Jia-Chen Gu, Zhen-Hua Ling, and Quan Liu. 2020. Utterance-to-Utterance Interactive Matching Network for Multi-Turn Response Selection in Retrieval-Based Chatbots. *IEEE ACM Trans. Audio Speech Lang. Process.* 28 (2020), 369–379. https://doi.org/10.1109/TASLP.2019.2955290

[17] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. 2017. 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*, Martha Palmer, Rebecca Hwa, and Sebastian Riedel (Eds.). Association for Computational Linguistics, 1049–1058. https://doi.org/10.18653/v1/d17-1110

[18] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2021), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

[19] Sadegh Kharazmi, Falk Scholer, David Vallet, and Mark Sanderson. 2016. Examining Additivity and Weak Baselines. *ACM Trans. Inf. Syst.* 34, 4 (2016), 23:1–23:18. https://doi.org/10.1145/2882782

[20] Antonios Minas Krasakis, Mohammad Aliannejadi, Nikos Voskarides, and Evangelos Kanoulas. 2020. 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*, Krisztian Balog, Vinay Setty, Christina Lioma, Yiqun Liu, Min Zhang, and Klaus Berberich (Eds.). ACM, 129–132. https://doi.org/10.1145/3409256.3409817

[21] Juntao Li, Chang Liu, Chongyang Tao, Zhangming Chan, Dongyan Zhao, Min Zhang, and Rui Yan. 2021. Dialogue History Matters! Personalized Response Selection in Multi-Turn Retrieval-Based Chatbots. *ACM Trans. Inf. Syst.* 39, 4 (2021), 45:1–45:25. https://doi.org/10.1145/3453183

[22] Yongqi Li, Wenjie Li, and Liqiang Nie. 2022. Dynamic Graph Reasoning for Conversational Open-Domain Question Answering. *ACM Trans. Inf. Syst.* 40, 4, Article 82 (jan 2022), 24 pages. https://doi.org/10.1145/3498557

[23] Jimmy Lin, Matt Crane, Andrew Trotman, Jamie Callan, Ishan Chattopadhyaya, John Foley, Grant Ingersoll, Craig MacDonald, and Sebastiano Vigna. 2016. 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 (Lecture Notes in Computer Science, Vol. 9626)*, Nicola Ferro, Fabio Crestani, Marie-Francine Moens, Josiane Mothe, Fabrizio Silvestri, Giorgio Maria Di Nunzio, Claudia Hauff, and Gianmaria Silvello (Eds.). Springer, 408–420. https://doi.org/10.1007/978-3-319-30671-1_30

[24] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. 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*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 2356–2362. https://doi.org/10.1145/3404835.3463238

[25] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2020. Distilling Dense Representations for Ranking using Tightly-Coupled Teachers. *CoRR* abs/2010.11386 (2020). arXiv:2010.11386 https://arxiv.org/abs/2010.11386

[26] Sheng-Chieh Lin, Jheng-Hong Yang, Rodrigo Nogueira, Ming-Feng Tsai, Chuan-Ju Wang, and Jimmy Lin. 2021. Multi-Stage Conversational Passage Retrieval: An Approach to Fusing Term Importance Estimation and Neural Query Rewriting. *ACM Trans. Inf. Syst.* 39, 4 (2021), 48:1–48:29. https://doi.org/10.1145/3446426

[27] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. 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*, Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer (Eds.). ACM, 1101–1104. https://doi.org/10.1145/3331184.3331317

[28] Craig Macdonald and Nicola Tonellotto. 2020. 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*, Krisztian Balog, Vinay Setty, Christina Lioma, Yiqun Liu, Min Zhang, and Klaus Berberich (Eds.). ACM, 161–168. https://doi.org/10.1145/3409256.3409829

[29] Ida Mele, Cristina Ioana Muntean, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, and Ophir Frieder. 2020. 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*, Jimmy X. Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (Eds.). ACM, 2057–2060. https://doi.org/10.1145/3397271.3401268

[30] Ida Mele, Cristina Ioana Muntean, Franco Maria Nardini, R. Perego, Nicola Tonellotto, and Ophir Frieder. 2021. Adaptive utterance rewriting for conversational search. *Inf. Process. Manag.* 58 (2021), 102682.

[31] Rodrigo Frassetto Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *CoRR* abs/1901.04085 (2019). arXiv:1901.04085 http://arxiv.org/abs/1901.04085

[32] Shon Otmazgin, Arie Cattan, and Yoav Goldberg. 2022. F-coref: Fast, Accurate and Easy to Use Coreference Resolution. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing, AACL/IJCNLP 2022 - System Demostrations, Taipei, Taiwan, November 20 - 23, 2022*. Association for Computational Linguistics, 48–56. https://aclanthology.org/2022.aacl-demo.6

[33] Shon Otmazgin, Arie Cattan, and Yoav Goldberg. 2022. LingMess: Linguistically Informed Multi Expert Scorers for Coreference Resolution. *CoRR* abs/2205.12644 (2022). https://doi.org/10.48550/arXiv.2205.12644 arXiv:2205.12644

[34] Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Douglas Johnson. 2005. 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 (Lecture Notes in Computer Science, Vol. 3408), David E. Losada and Juan M. Fernández-Luna (Eds.). Springer, 517–519. https://doi.org/10.1007/978-3-540-31865-1_37

[35] Gustavo Penha and Claudia Hauff. 2020. 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 (CEUR Workshop Proceedings, Vol. 2666)*, Giuseppe Di Fabbrizio, Surya Kallumadi, Utkarsh Porwal, and Thrivikrama Taula (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-2666/KDD_Converse20_paper_5.pdf

[36] Filip Radlinski and Nick Craswell. 2017. A Theoretical Framework for Conversational Search. In *Proc. CHIIR*. ACM, New York, NY, USA, 117–126.

[37] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. http://jmlr.org/papers/v21/20-074.html

[38] Gonçalo Raposo, Rui Ribeiro, Bruno Martins, and Luísa Coheur. 2022. 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 (Lecture Notes in Computer Science, Vol. 13186)*, Matthias Hagen, Suzan Verberne, Craig Macdonald, Christin Seifert, Krisztian Balog, Kjetil Nørvåg, and Vinay Setty (Eds.). Springer, 199–206. https://doi.org/10.1007/978-3-030-99739-7_23

[39] Stephen E. Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (2009), 333–389. https://doi.org/10.1561/1500000019

[40] Gerard Salton and Chris Buckley. 1988. Term-Weighting Approaches in Automatic Text Retrieval. *Inf. Process. Manag.* 24, 5 (1988), 513–523.

[41] Chongyang Tao, Wei Wu, Can Xu, Wenpeng Hu, Dongyan Zhao, and Rui Yan. 2019. 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*, J. Shane Culpepper, Alistair Moffat, Paul N. Bennett, and Kristina Lerman (Eds.). ACM, 267–275. https://doi.org/10.1145/3289600.3290985

[42] Svitlana Vakulenko, Shayne Longpre, Zhucheng Tu, and Raviteja Anantha. 2021. 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*, Liane Lewin-Eytan, David Carmel, Elad Yom-Tov, Eugene Agichtein, and Evgeniy Gabrilovich (Eds.). ACM, 355–363. https://doi.org/10.1145/3437963.3441748

[43] Nikos Voskarides, Dan Li, Pengjie Ren, Evangelos Kanoulas, and Maarten de Rijke. 2020. *Query Resolution for Conversational Search with Limited Supervision.* Association for Computing Machinery, New York, NY, USA, 921–930. https://doi.org/10.1145/3397271.3401130

[44] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. 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*, Qun Liu and David Schlangen (Eds.). Association for Computational Linguistics, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

[45] Yu Wu, Wei Wu, Chen Xing, Ming Zhou, and Zhoujun Li. 2017. 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*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, 496–505. https://doi.org/10.18653/v1/P17-1046

[46] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. 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*, Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White (Eds.). ACM, 55–64. https://doi.org/10.1145/3077136.3080809

[47] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. 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. https://openreview.net/forum?id=zeFrfgyZln

[48] Rui Yan. 2018. "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*, Jérôme Lang (Ed.). ijcai.org, 5520–5526. https://doi.org/10.24963/ijcai.2018/778

[49] Jheng-Hong Yang, Sheng-Chieh Lin, Chuan-Ju Wang, Jimmy J. Lin, and Ming-Feng Tsai. 2019. Query and Answer Expansion from Conversation History. In *TREC*.

[50] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. 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*, Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White (Eds.). ACM, 1253–1256. https://doi.org/10.1145/3077136.3080721

[51] Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. *ACM J. Data Inf. Qual.* 10, 4 (2018), 16:1–16:20. https://doi.org/10.1145/3239571

[52] Wei Yang, Kuang Lu, Peilin Yang, and Jimmy Lin. 2019. 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*, Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer (Eds.). ACM, 1129–1132. https://doi.org/10.1145/3331184.3331340

[53] Zhou Yang, Qingfeng Lan, Jiafeng Guo, Yixing Fan, Xiaofei Zhu, Yanyan Lan, Yue Wang, and Xueqi Cheng. 2018. 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 (Lecture Notes in Computer Science, Vol. 11168)*, Shichao Zhang, Tie-Yan Liu, Xianxian Li, Jiafeng Guo, and Chenliang Li (Eds.). Springer, 16–27. https://doi.org/10.1007/978-3-030-01012-6_2

[54] Shi Yu, Zhenghao Liu, Chenyan Xiong, Tao Feng, and Zhiyuan Liu. 2021. 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*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 829–838. https://doi.org/10.1145/3404835.3462856

[55] Zhou Yu, Ziyu Xu, Alan W. Black, and Alexander I. Rudnicky. 2016. 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 Linguistics, 404–412. https://doi.org/10.18653/v1/w16-3649

[56] Hamed Zamani, Johanne R Trippas, Jeff Dalton, and Filip Radlinski. 2022. Conversational Information Seeking. An Introduction to Conversational Search, Recommendation, and Question Answering. *arXiv.org, Information Retrieval (cs.IR)* arXiv:2201.08808 (January 2022).

[57] Cheng Xiang Zhai. 2008. Statistical Language Models for Information Retrieval: A Critical Review. *Found. Trends Inf. Retr.* 2, 3 (2008), 137–213. https://doi.org/10.1561/1500000008

[58] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. 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*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 1503–1512. https://doi.org/10.1145/3404835.3462880

[59] Edwin Zhang, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, Rodrigo Frassetto Nogueira, and Jimmy Lin. 2021. Chatty Goose: A Python Framework for Conversational Search. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 2521–2525. https://doi.org/10.1145/3404835.3462782