UNIVERSITY OF CALIFORNIA
RIVERSIDE

Efficient Inference in Open Retrieval Question Answering Systems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Muhammad Shihab Rashid

June 2024

Dissertation Committee:

Dr. Evangelos Christidis, Chairperson
Dr. Evangelos Papalexakis
Dr. Paea LePendu
Dr. Yue Dong

The Dissertation of Muhammad Shihab Rashid is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I am grateful to my advisor, collaborators, labmates, and my wife, without whose help, I would not have been here.

To my parents for all the support.

ABSTRACT OF THE DISSERTATION

Efficient Inference in Open Retrieval Question Answering Systems

by

Muhammad Shihab Rashid

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2024
Dr. Evangelos Christidis, Chairperson

With the latest advances in conversational agents like Siri and Alexa, and Large Language Models (LLMs) like ChatGPT and PaLM, Question Answering (QA) systems have become more important. Users submit millions of queries per day and it is up to the system to provide reliable, to-the-point answers. In this dissertation, we explore various aspects to improve such QA systems.

First, we tackle the problem of collecting high-quality training data for QA systems. We especially focus on public frequently asked questions (FAQ) data on the Web. FAQ chatbots rely on good quality FAQ data but there is no good source of FAQ data available and collecting them is a tedious task. Given the plethora of such question-answer pairs on the Web, there is an opportunity to automatically build large FAQ collections for any domain. Automatically identifying and extracting such high-utility question-answer pairs is a challenging endeavor, which has been tackled by little research work. Although identifying general, self-contained FAQs may seem like a straightforward binary classification problem, the limited availability of training data for this task and the countless domains make building machine learning models challenging. We propose QuAX: a framework for automatically extracting high-utility (i.e., general and self-contained) domain-specific FAQ lists from the Web. QuAX receives a set of keywords from a user and works in a pipelined fashion to find relevant web pages and extract general and self-contained questions-answer pairs.

Second, it is challenging for open retrieval conversational QA (OrConvQA) to model the history of a user conversation, to better answer the last user question. State-of-the-art OrConvQA systems use the same history modeling for all three modules (Retriever, Reranker, Reader) of the pipeline. We hypothesize this as suboptimal. Specifically, we argue that a broader context is needed in the first modules of the pipeline to not miss relevant documents, while a narrower context is needed in the last modules to identify the exact answer span. We propose NORMY, the first unsupervised non-uniform history modeling pipeline that generates the best conversational history for each module. We further propose a novel Retriever for NORMY, which employs keyphrase extraction on the conversation history, and leverages passages retrieved in previous turns as additional context.

Third, with the prevalence of powerful LLMs, LLM-based Reranker modules need to process a large number of passages to re-rank them given a query. However, LLM APIs can be very expensive (especially ChatGPT, GPT-4, etc.). We propose EcoRank, a budget-constrained LLM-based passage re-ranker that intelligently chooses which passages to spend the budget on, with what prompt strategy, and with which LLM API, within the given budget. We design an LLM cascading pipeline with a mixture of cheaper and expensive APIs that achieves the best performance within the given budget.

Fourth, we focus on the Retriever component of the QA system. Retrieval becomes particularly challenging if the document corpus is not available or indexed locally and is accessed via APIs. For example, legal document retrieval systems like PACER, LexisNexis, etc. charge a fee for retrieving each document. We argue that to improve the retrieval accuracy, we need to expand by leveraging both feedback from already retrieved relevant documents and LLMs. We propose ProQE, a progressive query expansion algorithm that iteratively expands the query by retrieving documents, evaluating them, and updating the weights of the expanded terms using our novel scoring function.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the latest advances in conversational agents like Siri, Alexa, and Large language models like ChatGPT, question answering systems are on the rise. Everyday, millions of queries are submitted and it is up to the system to provide factual and to the point answers. QA systems usually comprise of Retrieval Augmented Generation (RAG) structure, where the answers are generated from retrieved documents given the query. There are three components to a RAG pipeline, 1) the Retriever, which retrieves top-k documents given a query. The second module 2) the Rearanker re-ranks these retrieved documents as the retriever component may not work perfectly. As there are millions of documents on the web, many such documents may seem relevant given the query. The reranker makes sure the top few passages are relevant. The final component 3) the Reader either extracts or generates the answer from the top few documents.

In this dissertation, we try to improve the training and inference process of such RAG systems. First, we tackle the problem of collecting high-utility training data. In frequently asked questions (FAQ) we see a scarcity of good quality training data. A data point which is a question-answer pair is considered high quality if the question is a *general* question, meaning that the question should be applicable for all domains and not only for specific domains. Second, the question should be a *self-contained* question, meaning, there should not be any co-references or ellipses. We propose a novel six-staged pipeline QuAX, which automatically finds QA pairs on the web and classifies them to be either general or self-contained. Our experimental results show that QuAX performs significantly more than baselines.

Second, we focus on the conversational question answering problem. In a typical OrConvQA pipeline, recent works use the same history modeling in all three modules of the pipeline. We hypothesize this as suboptimal. Specifically, we argue that a broader context is required at the earlier stages of the pipeline and as we move towards the right, the historical context should get narrower. We propose the first unsupervised non-uniform pipeline NORMY that selects the best conversational history to model. Our experimental results show that NORMY outperforms other SOTA baselines. We also publish a new dataset called doc2dial-OR to facilitate research for OrConvQA.

Third, we focus on the second module of the pipeline: the Reranker. Recently, LLMs have been used as text re-rankers and have been shown to perform well. However, LLMs can be very expensive. Many LLMs are available via APIs, which are hosted by commercial organizations. They charge a fee for using such APIs. In text re-ranking, this cost becomes particularly important as businesses need to rerank hundreds of documents per query every day. As they deal with thousands of queries each day, *budget* becomes their most essential concern. In this chapter, we introduce the problem of budget-aware text-reranking and propose a suit of budget-constrained models. Our most efficient model called EcoRank is a two-layer cascading pipeline that jointly optimizes several key decisions: which prompts to apply, which LLM APIs to use, and how to split the budget between prompts. Our experimental results show that EcoRank performs better than other strong baselines.

Finally, we focus on the first module of the pipeline, which is the Retriever. A good retriever is very important to retrieve good quality passages. Query expansion is a popular method to better retrieve documents. Traditional query expansion methods use pseudo-relevance feedback to find good keywords to append to the query. However, if the retrieved passages are not relevant, the expansion terms become noisy. To combat this, recent studies use LLMs to generate additional content as expansion terms. LLMs are also prone to hallucinations which may hinder the effectiveness of this approach. Further, all these work saliently assume that majority of the cost is associated with the LLMs and almost no cost is associated with the Retriever. We argue that this is not true for several important problem settings. Especially, when the corpus is not indexed locally and is only available via APIs, like in different law firms PCER, WestLaw, etc. In real life scenarios, the dominant cost is the retriever cost. We argue that an effective retriever should be cost-effective and should utilize the feedback from both retrieved passages and LLMs. We propose ProQE, which is a progressive query expansion algorithm that iteratively retrieves passages and expands terms. Our novel weight function puts weights on the expansion terms from LLM feedback. Further, ProQE is plug-and-play, meaning it can work with any type of sparse or dense retrieval method.

Chapter 2

Mining the Web for High-utility FAQs

2.1 Introduction

Frequently Asked Questions (FAQ) lists provide users with frequently requested information on a given topic. For example, many healthcare providers offer a FAQ list on COVID-19, which allows users to obtain relevant information with ease. More importantly, FAQ lists also facilitate many important tasks such as retrieval-based question answering [105], training generative question answering models [115], augmenting chatbot knowledge bases [70], and allowing search engines to provide a short, relevant list of question-answer pairs when given a search query [106].

[1]Based on a study of 1,176 questions extracted from 170 FAQ web pages in the mental counseling and dental health domains.

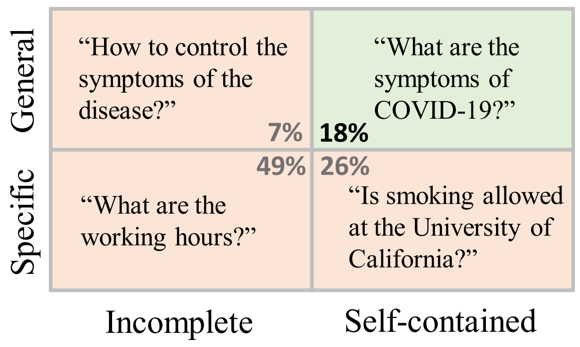

General
Specific
"How to control the symptoms of the disease?"
7%
"What are the symptoms of COVID-19?"
18%
49%
26%
"What are the working hours?"
"Is smoking allowed at the University of California?"
Incomplete
Self-contained

Figure 2.1: Only 18% [1] of FAQ on the Web are general and self-contained; these are the only questions that are suitable for building general-purpose knowledge-bases.

Reliable FAQ lists are typically created and maintained manually by domain experts, which is laborious and time consuming. The Web offers a plethora of FAQ lists on almost every topic; thus, mining the Web for FAQ lists provides a scalable way of acquiring and curating FAQs.

Automatically mining and curating FAQ lists from the Web is a challenging task due to the different ways of presenting FAQ lists on the Web, and the inherent noisiness of the available FAQs. There have been many works on mining FAQ from the Web; for example, the authors in [87] propose extracting FAQ lists from web pages by identifying HTML list constructs in web pages and the authors in [46] mine online forums for question-answer pairs using sequential pattern features and graph-based ranking. Existing FAQ mining works [14] focus on retrieving FAQ lists, and they neglect filtering out noisy question-answer pairs.

Although the mere retrieval of large-scale FAQ lists is useful on its own, low utility FAQ lists may provide incomplete or misleading information. Specifically, for question-

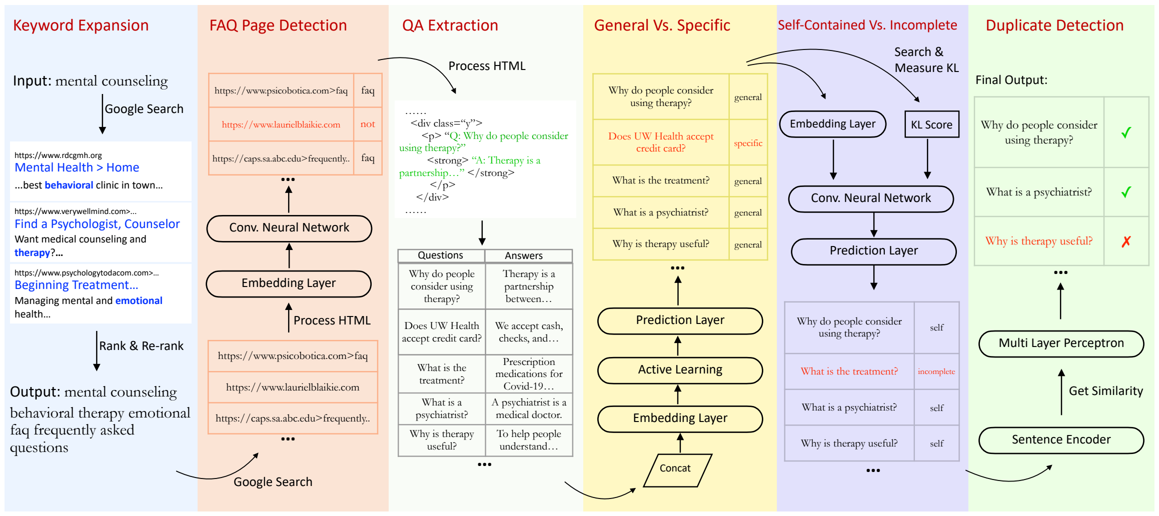

Keyword Expansion
Input: mental counseling
Google Search
https://www.rdcgmh.org
Mental Health > Home
...best behavioral clinic in town...
https://www.verywellmind.com>...
Find a Psychologist, Counselor
Want medical counseling and therapy?...
https://www.psychologytodacom.com>...
Beginning Treatment...
Managing mental and emotional health...
Rank & Re-rank
Output: mental counseling behavioral therapy emotional faq frequently asked questions
FAQ Page Detection
https://www.psicobotica.com>faq
faq
https://www.laurielblaikie.com
not
https://caps.sa.abc.edu>frequently..
faq
Conv. Neural Network
Embedding Layer
Process HTML
https://www.psicobotica.com>faq
https://www.laurielblaikie.com
https://caps.sa.abc.edu>frequently..
Google Search
QA Extraction
Process HTML
<div class="y">
<p> "Q: Why do people consider using therapy?"
<strong> "A: Therapy is a partnership..." </strong>
</p>
</div>
Questions
Answers
Why do people consider using therapy?
Therapy is a partnership between...
Does UW Health accept credit card?
We accept cash, checks, and...
What is the treatment?
Prescription medications for Covid-19...
What is a psychiatrist?
A psychiatrist is a medical doctor.
Why is therapy useful?
To help people understand...
General Vs. Specific
Why do people consider using therapy?
general
Does UW Health accept credit card?
specific
What is the treatment?
general
What is a psychiatrist?
general
Why is therapy useful?
general
Prediction Layer
Active Learning
Embedding Layer
Concat
Self-Contained Vs. Incomplete
Search & Measure KL
Embedding Layer
KL Score
Conv. Neural Network
Prediction Layer
Why do people consider using therapy?
self
What is the treatment?
incomplete
What is a psychiatrist?
self
Why is therapy useful?
self
Duplicate Detection
Final Output:
Why do people consider using therapy?
✓
What is a psychiatrist?
✓
Why is therapy useful?
✗
Multi Layer Perceptron
Get Similarity
Sentence Encoder

Figure 2.2: Overview of QuAX.

answer pairs to be useful to a wide audience, outside the Web site where the question is hosted, the questions must be **general** and **self-contained**. General questions are those whose utility is universal. For example, the question *"What are the symptoms of COVID-19?"* asks for information that is universally applicable, and thus is of high utility. On the the hand, questions such as *"Is smoking allowed at the University of California?"* appear in contexts where users ask for information that pertain to a certain entity; such questions have limited utility and thus should not be included in general-purpose FAQ lists (of course this question is useful for students of the University of California, but our goal is to extract questions with much wider scope). Self-contained questions are ones that are complete on their own, in the sense that they do not contain references or ellipses. In contrast, questions such as *"How to control the symptoms of the disease?"* require access to a larger context than the question-answer pair to be useful; consequently, such questions should also be discarded when building universal FAQ lists.

It turns out that only a small percentage of FAQ on the web is general and self-contained and thus are useful for general-purpose knowledge-bases (see statistics in Figure 2.1). Identifying these questions is challenging because they do not follow a specific pattern; hence, using static rules to identify such questions is not feasible, and the lack of high-utility labeled training data makes it difficult to train a classification model. Furthermore, if machine learning models are to be used, such models should be domain-oblivious. A key requirement of QuAX is that no domain-specific training data should be required.

We propose QuAX: a framework for retrieving general and self-contained FAQ lists from the Web for a given domain. Figure 2.2 shows an overview of QuAX, which receives a list of keywords that describe a certain domain (e.g. COVID-19 or Plastic Surgery), and works in six pipelined steps to produce a high-utility list of FAQ on the given domain as follows. First, QuAX augments the given list of keywords to include extra terms that would help retrieve comprehensive yet relevant FAQ pages. Then, the expanded list of keywords is used to make a Google search to retrieve web pages with relevant information to the given domain. The retrieved web pages are fed into our FAQ Page Detection module, which first pre-processes the HTML content of the pages and then uses a CNN based classifier to identify pages that contain FAQ lists. After that, our QA Extraction module uses HTML tags to extract the actual question-answer pairs from the given FAQ pages. Our General vs. Specific module filters out specific questions using a CNN classifier and an active learning strategy to mitigate the scarcity of training data. Our Self-contained vs. Incomplete module then filters out incomplete questions using a CNN classifier coupled with a KL-divergence based feature generator. Finally, our Duplicate Detection module filters out redundant questions using a multilayer perceptron. To train our classifiers, we propose a strategy to collect and annotate reliable training examples.

We evaluate each module in our pipeline and compare its performance against that of strong baselines and show that each of our individual modules outperforms the baselines. Furthermore, we perform case studies with five domains: mental counseling, dental health, plastic surgery, medical marijuana, and COVID-19. For each domain, we generate a set of descriptive keywords and pass them through our pipeline, and we show that the resulting FAQ lists are relevant, general, and self-contained.

In summary, we claim the following contributions in this chapter:

- We propose the first complete pipeline for retrieving comprehensive yet high-utility, general and self-contained, FAQ lists.
- We collect and annotate training data for training the classifiers in our modules, published on a public repository.
- We show that an active learning strategy and a KL-divergence based feature extraction method help mitigate the scarcity of training data in our most critical modules (the General vs. Specific, and Self-contained vs. Incomplete modules).
- We experimentally evaluate our pipeline and show that our resulting FAQ lists are of high utility.

2.2 QuAX System

In this section we describe each module of the QuAX pipeline shown in Figure 2.2.

2.2.1 Keyword Expansion Module

To extract high-utility FAQs, the mined web pages must contain question-answer pairs that are relevant to the input keywords. Due to the high volume of content on the web, selecting the right set of keywords can be challenging, this problem is known as *search keyword mining* [163]. Using such keywords to search the web for relevant pages may result in retrieving pages that may not contain question-answer pairs, or that are irrelevant to the input keywords due to users not having deeper domain knowledge when selecting initial keywords.

Existing works on keyword expansion [25] enable producing extra keywords that can improve the relevance of the retrieved pages (such as retrieving relevant twitter posts). However, such works do not particularly produce keyword expansions that result in retrieving pages with question-answer pairs. In our keyword expansion module, we extend the work in [163] to produce keyword expansions that not only facilitate retrieving more relevant pages but also contain question-answer pairs.

The authors in [163] present a keyword expansion algorithm that retrieves more relevant twitter posts by using a double ranking approach. However, it does not take into consideration the type of content extracted (i.e. faqs). We extend their work as follows: First, it retrieves web pages using the domain input keywords and ranks the words in the first k retrieved pages based on their entropy. Then, our module uses the l top-ranking words in the resulting vocabulary to do another search. The words in the newly retrieved paged are ranked again based on their entropy and the the l top-ranking ones in the re-ranked list are returned as keywords expansions. We repeat these steps for the word "faq" separately and concatenate the resulting words with the initial result. The system may return very common words (e.g., *the* and *of*) from the English vocabulary. To mitigate this, similarly to [163], we use a Random Words Set (RS), consisting of 400k words, where the words are taken from 300 random Wikipedia articles. If any word appears frequently in the random set, it gets a lower entropy score. We use the following formula to compute entropy:

$$e_{\mathrm{w}} = -\sum_{s} \frac{f_{\mathrm{s}}(w) + \lambda}{\sum_{\mathrm{s}} f_{\mathrm{s}}(w) + |S|\lambda} log_2 \frac{f_{\mathrm{s}}(w) + \lambda}{\sum_{\mathrm{s}} f_{\mathrm{s}}(w) + |S|\lambda}, \tag{2.1}$$

where s $\in$ S = {SS, RS}, *SS* being the Snippets Set (Snippets returned from searching Google), *RS* being Random Set. λ and —S— are smoothing parameters in case any word does not appear in any snippet. They are set to 0.005. Snippet frequency (how many times a word appears in a snippet) is denoted by *f(w)*.

2.2.2 FAQ Page Detection Module

Since most modern web pages are dynamic, their complex DOM structure makes classification and information extraction challenging. To overcome this, our FAQ Page Detection Module first converts dynamic pages into static ones by flattening interactive elements in a page into a single-layer DOM tree using the boiler-pipe APIs [2] article extractor. Our module then uses a Convolutional Neural Network (CNN) model with HTML pre-processing to classify the resulting static HTML pages into FAQ or NOT-FAQ pages. We

[2] https://boilerpipe-web.appspot.com/

chose a CNN due to their reported success in text classification tasks [81] and having the best performance among baseline deep learning techniques. First, our module pre-processes input HTML pages such that HTML tags which do not usually contain question answer pairs are replaced with uniform labels (For example, `<div>` and `<a>` are replaced with `TAG1`) and tags which contain the question answer strings (`<h1-4>, <p>`) are replaced with `TAG2`, and questions marks are replaced with the tag `QUESTION MARK`. As we we show in our experimental evaluation, this pre-processing results in improved classification accuracy. Given the pre-processed pages, our model generates an embedding for each word in an input HTML page using a pre-trained word2vec [110] and then combines these embeddings into a feature matrix. Our embedding layer is connected to 5 parallel one dimensional convolutional layers with a filter size of 200 and ReLU activation. We use a global max pooling layer to reduce the size of the feature map and a Sigmoid activation function (to accommodate our classification task) at our last dense layer.

We train our model using training data generated as follows. To generate positive examples, we use the Google search results for keywords from different domains concatenated with the word 'FAQ' and use the first 25 pages. We generate negative examples similarly but with without adding the 'FAQ' keyword. We train our model using the Adam optimizer with binary cross-entropy loss function.

2.2.3 QA Extract Module

Although there are tools for extracting question-answer pairs from FAQ pages, some of these tools are proprietary [14] and the others require lots of labeled training

Algorithm 1 QA Extractor

```
Ensure: UQA: Unclassified QA Pairs
1: for each website $w \in FW$ do
2:   $D_w \leftarrow$ ExtractHTML($w$) {Extract DOM structure of web page}
3:   if Check1Catg($D_w$) is true {Web page falls into category 1} then
4:     for each element $e \in D_w$ do
5:       if element $e \in \{h1, h2, h3, h4\}$ then
6:         $NH_e \leftarrow$ ExtractNextElement($e$) {Parse next element in DOM tree}
7:         if $NH_e \in \{p, div\}$ and length($e$) $>$ $min\_Qlength$ and length($NH_e >$
           $min\_Alength$) then
8:           $Q \leftarrow Q \bigcup$ Text($e$), $A \leftarrow A \bigcup$ Text($NH_e$)
9:   else
10:    if Check2Catg($D_w$) is true {Web page falls into category 2} then
11:      Repeat 5-10 for <p> or <div> pair
12:    else
13:      for each element $e \in D_w$ do
14:        $C_e \leftarrow$ GetAllChildElements($e$)
15:        for each child element $c \in C_e$ do
16:          if $e \in \{p, div\}$ and $c \in \{strong, br, a\}$ then
17:            $Q \leftarrow Q \bigcup$ Text($e$), $A \leftarrow A \bigcup$ Text($c$)
18:            $UQA \leftarrow UQA \bigcup \{Q, A\}$
19: return $UQA$
```

data [75]. Therefore, we built our own algorithm for this task. Our question-answer extraction module is based on Algorithm 1. Our algorithm utilizes the HTML tree structure of pages; the main insight is that question-answer pairs are usually nested within certain HTML tags such as `<h>`, `<p>`, and `<div>`.

Algorithm 1 receives a list of FAQ pages as input, and it produces a list of question-answer pairs. For each page, Algorithm 1 works as follows.

Variable Q and A stores questions and answers extracted from the webpage (line 1). We extract the HTML tree of the webpage using Jsoup and store in D_w (line 3). The HTML is cleaned using boiler-pipe. Each static cleaned FAQ webpage is usually divided into following three categories. 1) The questions are in `<h>` tag and answers are in either `<p>` or `<div>`. 2) Questions and answers are in different `<p>` or `<div>`. 3) Questions and answers are both in same `<p>` or `<div>`. Line 4 checks whether the webpage falls in category 1. If yes, then Line 5-8 checks each element in HTML tree, if its a <h> tag then it looks at the next element of tree (line 7). If the next element is a `<p>` or `<div>` and the text length of the tag is greater than certain threshold, we add the textual content of the tags in corresponding Q and A (line 9-10). The intuition being, if a question resides in a `<h>` block, the subsequent block should contain textual contain which may be considered as the answer to that question.

If the webpage falls in category 2 (line 12), which means the question does not reside in `<h>` block, then we repeat the same process as before for `<p>` and `<div>`. The extraction process becomes trickier when both the question and answer is situated under the same tag. Usually the question is separated from the answer using tags like `<strong>`, `<br>` etc. So we check all the child elements (line 18) and if child element contains `<br>` or `<strong>` we put the textual content of parent tag in Q and content of child tag in A.

2.2.4 General vs. Specific Classification

This module identifies general questions given a list of question-answer pairs that include both general and specific questions. Although this is a straightforward binary text classification task, the scarcity of training data makes it challenging. Furthermore, the training data for such a text classifier shall be domain-oblivious for the module to accommodate any domain. We mitigate the training data scarcity issue using active-learning [16, 22, 170] and we select our training data carefully in such a way that our active-learning classifier is kept domain-oblivious. We describe our active-learning classifier followed by our data collection methodology in the rest of this section.

Active-learning has shown good results in text classification tasks where training data is scarce [16,22]. It is a form of semi-supervised learning that uses self-learning feature. This technique first learns from a standard automated labeled training data then continues to learn labels from domain specific unlabeled data that it infers with high confidence. The predicted unlabeled instances with high confidence are added to the standard model and are re-trained. The intuition is that such labels resemble human-labeled data and thus allow the classifier to provide better predictions for data points whose inferred labels' confidence is not conclusive. We select the training data points among the unlabeled instances using *uncertainty sampling* using modAL framework [52]. This sampling technique uses the posterior probabilities of the resulting labels produced by a model θ to select the labels with the

Table 2.1: QuAX datasets summary

Module	Num. of examples	Avg. tokens/instance	Description
FAQ Page detection*	250 websites	1,862	HTML of webpages
General vs. Specific*	19,442 sentences	16	Mayo Clinic & Twitter
Self-cont. vs. Incomplete*	1,002 questions	8	FAQ from the web
Duplicate Detection†	404,291 question pairs	23	Labeled question pairs

*Original datasets; Available at: https://github.com/shihabrashid-ucr/quax-dataset

†Available at: www.kaggle.com/c/quora-question-pairs.

highest levels of confidence. We use the equation below to calculate posterior probabilities:

$$\phi_{\text{LC}}(x) = \underset{x}{argmax}(1 - P_\theta(\hat{y}|x)) \tag{2.2}$$

where x is the instance to be predicted and $\hat{y}$ is the most likely prediction.

To train our base model in this module, we collect training sentences that are general (i.e., have few instances of ellipsis and co-reference). We collect such data by extracting random sentences from Mayoclinic articles [3]. We collect Specific sentences by selecting responses to specific user issues from Twitter customer support dataset [4]. Such sentences are specific because they address issues that pertain to specific entities.

2.2.5 Self-contained vs. Incomplete Classification

Given a list of general questions, this module extracts the ones that are self-contained. We use a CNN classifier and we propose a novel multi-feature extraction method with KL that is designed to improve our classifier's ability to distinguish self-contained

[3]https://www.mayoclinic.org/

[4]https://www.kaggle.com/thoughtvector/customer-support-on-twitter

questions. We use the intuition that self-contained questions (i.e., *"Can I get COVID-19 from my pets?"*) can be answered without knowing any context, which means that the answers to such questions have a high degree of similarity. We retrieve answers to a given question using the Google search engine: we issue the question as a query and consider the first ten snippets as answers, given that these typically provide direct or closely related answers. We quantify the similarity across answers by calculating the Kullback-Leibler divergence score [85] (KL) using the following equation:

$$D_{\mathrm{KL}}(P||Q) = -\sum_{x \in \chi} P(x) log(\frac{Q(x)}{P(x)}), \tag{2.3}$$

where P and Q are defined over the same probability space χ.

KL divergence is a statistic used to measure the similarity between two probability distributions and it is typically used in information retrieval to measure similarity across documents. Here, the probability space χ represents all words occurring in the union of two lists of snippets. We use term frequencies of each word to calculate the probability of a word ($P(x)$) given a snippet. We compute the average pair-wise KL divergence score for the snippets that answer a question and pass the floating point KL score to our classifier as a feature in addition to the word embeddings of the input query. We train our model using manually labeled datasets from five domains. We labeled 200 questions from each domain, resulting in a total of 1000 training examples. While training, we do not use instances from the same domain (i.e. we use 800 examples for training for a particular domain). We use domain-specific data in inference time, ensuring fairness and domain independence.

2.2.6 Duplicate Detection

Questions such as *"What is rhinoplasty?"* and *"How do you define rhinoplasty?"* are equivalent despite being expressed differently. To produce a higher utility list of question-answer pairs, we eliminate duplicate questions, where duplicates include questions that are semantically very similar. Since we need to identify semantically similar questions even if they have a large string-based distance, record linkage [57] and other string-based methods are inadequate. Instead, we build a similarity classifier which we train using a question similarity dataset from a Kaggle duplicate detection competition (dataset details are in Section 3.4). We use the Google universal sentence encoder to encode our questions before passing them to a sequential multi layer neural network. The input to the neural network model are question pairs, which are put through Google sentence embedder of dimension 512. The encodings are concatenated and batch normalized to avoid overfitting. ReLU activation function and "adam" optimizer are used. For each pair of questions, the output is a binary 0 or 1 which indicates whether the pair is a duplicate or not.

2.3 Experimental Evaluation

We evaluated our framework on five domains in the healthcare area: mental counseling, dental health, plastic surgery, medical marijuana, and COVID-19. We start with simple keywords that describe each respective domain (i.e., the domain name succeeded by the word "faq"), for example "mental counseling faq", and obtain a list of FAQs using our framework. Along the way, we evaluate each component independently. We present our experimental results in the subsequent sections and show that our framework produces a list of high-utility FAQs (i.e., general and self-contained question-answer pairs) and that its modules provide more accurate results than strong baselines that could have been used in our modules' place. Since the training data and the evaluation metrics for each module are different, we present these in each respective subsection.

Datasets. To fully automate high-utility FAQ extraction and tackle the training data scarcity in the context of our trainable modules, we have created three training datasets for the modules FAQ page detection, General vs. Specific classification, and Self-contained vs. Incomplete classification, respectively. For the rest of our trainable modules, we have used publicly available datasets. Table 2.1 summarizes the datasets we used to train and evaluate our trainable modules. All examples in the datasets of the FAQ Page Detection and Self-contained vs. Incomplete modules are labeled manually by annotators. For the General vs. Specific dataset, the examples are automatically labeled. The original datasets we have collected and annotated can facilitate further research on high-utility FAQ extraction; we explain the procedure of collecting our original datasets in each respective subsection.

2.3.1 Keyword Expansion

Competitors. We compare our module to a simple baseline where we use the input keywords as is, and we also compare it against *Double Rank* [163] where the authors presented a re-ranking algorithm of keywords expansion based on entropy.

Evaluation Methodology. We use Precision, which is a standard evaluation metric in Information Retrieval, to evaluate our keyword expansion module. Precision is computed by dividing the total number of "correct" webpages returned by searching Google using the keywords by the total number of webpages returned.

We first search Google with expanded keywords and take into consideration the top 50 returned results from Google. If any returned webpage is a FAQ page and is on topic then it is a correct result. By "on-topic" we mean that the content of the webpage is related to the domain. We produce ground truth by manually evaluating whether each webpage is precise or not. Similarly to [163], for the values of k and l, we used 10 and 8, respectively.

Results. We present our results in Table 2.2. In the final column we show results of our module. We see that, in most cases, our method performs better than the baseline (Without Keyword Expansion (KE)) and *Double Ranking (DR).* Only for dental health, our system performs worse than both the baseline and *Double Ranking.* For this domain, the expanded keywords are "dental health hygiene teeth gums decay". We see that some of the keywords here are general and apply to many different domains related to healthcare like "hygiene" and "decay". Because of this, some of the returned results were articles regarding hygiene and not FAQ pages. Overall our updated algorithm performs better than both baseline and *Double Ranking*; our module achieves 17% higher precision on average.

2.3.2 FAQ Page Detection

Competitors. We show the performance of many deep learning-based classifiers including RNN [111], LSTM [67], BiLSTM [176], and Self-BiLSTM [179].

Table 2.2: Keywords precision

Domain	Without KE	DR	Updated DR
Mental Counseling	0.7000	0.7200	0.8000
Dental Health	0.8200	0.8600	0.7800
Plastic Surgery	0.8400	0.8000	0.9600
Medical Marijuana	0.6400	0.8400	0.7800
Covid 19	0.5200	0.6800	0.7800
Average	0.7000	0.7800	**0.8200**

Evaluation Methodology. For each domain, the test set includes the top 25 returned FAQ websites and the top 25 returned not-FAQ websites by searching Google with the expanded keywords. The training set is our proposed dataset which includes 250 labeled HTML pages. While training for a domain, the data points from that domain are omitted for fair results. We use accuracy and F1 scores in this evaluation.

Results. We see in Table 2.3 that CNN provides significantly better performance than the rest of the classifiers. The textual content of an HTML page is complex and large; therefore, convolutional networks can take the best advantage of such complex structure. We also see from Table 2.4 that our method achieves an average increase of 5% in accuracy and 6% increase in F1 scores.

2.3.3 QA Extraction

To evaluate this module, we collected 100 different FAQ websites using Google search and fed them to our algorithm to see whether it is able to extract QA pairs. The 100

Table 2.3: FAQ page detection accuracy (baselines)

Domain	Base (CNN)	Base (RNN)	Base (LSTM)	Base (BiLSTM)	Base (SelfBiLSTM)
MC	0.8400	0.6400	0.6200	0.6200	0.6200
DH	0.8800	0.6200	0.5800	0.6000	0.6400
PS	0.8400	0.6000	0.5800	0.5800	0.6200
MM	0.8800	0.6600	0.6000	0.6000	0.6000
CO	0.8600	0.6000	0.5800	0.6200	0.6000
Average	**0.8600**	0.6240	0.5920	0.6040	0.6160

web pages were taken from a mixture of the domains. Our algorithm was able to successfully extract 71 webpages out of 100.

2.3.4 General vs. Specific Classification

Competitors. We show the results for several deep learning based text classifiers (Table 2.5) and then show the effect of integrating our active learning approach in Table 2.6.

Evaluation Methodology. We use our 19,000 training data points to train our classifier in this module. For general training data set, we need textual content that are general in nature, meaning there are less number of ellipsis and co-reference and free from context. We propose Mayoclinic websites articles as the source for our training data for class: general. We extract 9,325 random sentences from articles chosen at random about different diseases, medicines from their website and label them as ”general”. Our training datasets are domain independent due to the nature of the sentences being used as data points. Finding ”specific” sentences was challenging because most documents on the web focus on general textual content. We chose the Twitter customer support dataset from

Table 2.4: FAQ page detection comparison

Domain	CNN		QuAX Page Detector	
	Accuracy	F1	Accuracy	F1
MC	0.8400	0.8400	0.9000	0.8900
DH	0.8800	0.8650	0.9400	0.9350
PS	0.8400	0.8350	0.8600	0.8500
MM	0.8800	0.8700	0.9200	0.9150
CO	0.8600	0.8500	0.9200	0.9150
Average	0.8600	0.8500	**0.9080**	**0.9010**

Kaggle and extracted the tweets sent by customer service agents of any specific organization and users filing any complaint. We took 10,117 tweets and labeled them as specific. We use an embedding layer which uses word2vec and projects each sentence into 300 dimensional vector. We test the model with our extracted QA pairs. We concatenate a question and its corresponding answer into one string and then test.

Results. We see from Table 2.6 that our active learning approach improves the performance of CNN (the best performing classifier). The main reason is that, in baseline, the training dataset does not hold too much information. Because the data points were automatically extracted and labeled, not all labeled data points can be accurate. Using a semi-supervised approach like active-learning mimics human-labeled data and thus produces better results. We observed better results if questions and answers are merged into a single strings (each pair). Our test set contains around 200 question-answer pairs for each domain.

Table 2.5: General vs. Specific classification accuracy (baselines)

Domain	Base (CNN)	Base (RNN)	Base (LSTM)	Base (BiLSTM)	Base (SelfBiLSTM)
MC	0.6670	0.6150	0.5567	0.5690	0.6280
DH	0.7343	0.6640	0.6610	0.6754	0.6670
PS	0.7230	0.6410	0.5830	0.5830	0.6230
MM	0.6338	0.5778	0.5319	0.5322	0.5715
CO	0.7215	0.6730	0.6020	0.6410	0.6678
Average	**0.6959**	0.6341	0.5869	0.6001	0.6314

2.3.5 Self-contained vs. Incomplete Classification

Competitors. We use supervised deep learning approaches as baselines and show their results in Table 2.7. We pick the best performing method and integrate our KL method to show its efficacy.

Evaluation Methodology. We use our proposed training dataset of 1000 manually labeled questions. To ensure fairness, while training for a domain, we do not include data points from that domain. This shows, our method is domain oblivious. For testing, we used ∼150 questions from each domain with equal class sizes.

Results. As seen from table 2.8, integrating our KL method into the best performing classifier CNN improves its performance across all domains. Note that even the best performing classifier struggles to achieve impressive accuracies and F1 scores because of the limited amount of training data used. However, our contribution shows that using a multi-feature approach can improve the performance of strong baselines while being domain independent.

Table 2.6: General vs. Specific classification comparison

Domain	CNN		Active Learning + CNN	
	Accuracy	F1	Accuracy	F1
MC	0.6670	0.6560	0.7118	0.7008
DH	0.7343	0.7128	0.7656	0.7474
PS	0.7230	0.7210	0.7830	0.7800
MM	0.6338	0.6042	0.7042	0.6913
CO	0.7215	0.6823	0.7974	0.7567
Average	0.6959	0.6752	**0.7524**	**0.7352**

2.3.6 Duplicate Detection

For duplicate detection, we use the 400,000 quora QA pairs dataset to train. Our duplicate detection module shows 80% accuracy while testing on with Quora dataset.

2.3.7 Entire Framework Evaluation

We present in this subsection the average precision and average recall for our end-to-end pipeline. To calculate precision for each domain, we divide the number of general and self-contained questions by the total number of QA pairs generated by the system. To calculate recall for each domain, we divide the total count of general and self-contained questions generated by duplicate detection module by the number of all general and self-contained questions generated by QA extraction module. Our system achieves an average precision of 78.6% and an average recall of 60.20%. Note that, the goal of a high utility FAQ extractor should be to not generate false positives. However, missing out on some FAQs, which results in a relatively low recall, should not be an issue considering huge number

Table 2.7: Self-contained vs. Incomplete classification accuracy (baselines)

Domain	Base (CNN)	Base (RNN)	Base (LSTM)	Base (BiLSTM)	Base (SelfBiLSTM)
MC	0.6500	0.5400	0.6000	0.5600	0.5900
DH	0.6230	0.6385	0.5901	0.5081	0.4918
PS	0.5990	0.5990	0.5545	0.6000	0.5545
MM	0.6640	0.6150	0.5983	0.5664	0.5664
CO	0.5324	0.5040	0.4748	0.5100	0.5539
Average	**0.6136**	0.5793	0.5635	0.5489	0.5513

of FAQ websites on the Web. These results show that our framework manages to obtain reasonable percentage of high-utility question-answer pairs despite training data scarcity.

2.3.8 Case Studies

We present in this subsection statistics and qualitative analysis of the results of our framework in the domains we have selected, and we further discuss a sample from the results in the mental counseling and the COVID-19 domains. Table 2.9 shows the counts of the results of each module in our framework. We used the top 100 results from Google and pushed them through our framework. We chose 100 results because Google API has a limit of 100 results per request. From this table we can make the following observations:

- The total number of general questions is low compared to the total number of QAs on the web. This shows the importance of classification of general questions to build knowledge base. It is not enough to extract QAs and use them as knowledge bases.

Table 2.8: Self-contained vs. Incomplete classification comparison

Domain	CNN		KL + CNN	
	Accuracy	F1	Accuracy	F1
MC	0.6500	0.6485	0.6853	0.6746
DH	0.6230	0.6012	0.6400	0.6370
PS	0.5990	0.5920	0.6363	0.6300
MM	0.6640	0.6520	0.6923	0.6811
CO	0.5325	0.5240	0.5600	0.5410
Average	0.6137	0.6035	**0.6427**	**0.6327**

Table 2.9: Performance of QuAX

Domain	No FAQ	Total QA	GQ Detected	SC Detected	w/o Duplicates	% of High-utility Qstns.
MC	83	776	363	223	219	70
DH	94	559	446	256	251	88
PS	92	783	518	284	277	79
MM	91	617	327	127	127	82
CO	82	855	679	285	282	74

- In each module, low-utility question-answer pairs are filtered out incrementally until reaching the last step where a list of high-utility question-answer pairs are produced.

- For the COVID-19 domain, the number of general questions detected is very high. This is because, most questions regarding COVID-19 are general as this is a recent topic. There are not as many individual organizations that have FAQs about COVID-19 compared to other domains.

Table 2.10: Top 10 Extracted Questions

Domain	Questions
Mental Counseling	Why do people consider using therapy?
	For what concerns do students seek personal counseling?
	What are the different types of mental health professionals?
	How long are therapy sessions themselves?
	What is psychotherapy?
	What behavioral health concerns does UW Health treat?
	How does a student know if s/he needs counseling?
	What are the Benefits of Telemental Health?
	What is the purpose of this website?
	What is genetic counseling?
Covid 19	How can you tell the difference between the novel coronavirus and a cold?
	What are the symptoms in children?
	What is social distancing?
	What does it mean that covid-19 is a global pandemic?
	What is the state recommending for social distancing?
	When are you open for vaccines?
	What are the treatments for covid-19?
	What is quarantine?
	I have been around someone else who was exposed to a person with covid-19. What should I do?
	Does health insurance cover covid-19 testing and care?

Table 2.10 shows examples of the generated questions for the mental counseling and the COVID-19 domains. From Table 2.10, in the mental counseling domain, we see that there are some questions that are not properly classified. For example, Question 6 *"What behavioral health concerns does UW health treat?"*. Although this is a self-contained question, it talks about information regarding University of Washingtons health system. This is a specific question but this type of question is difficult for the system to detect. The training

dataset does not have information regarding "UW" being a specific organization and thus it mistakenly classifies it to be a general word. We also see that Question 9 *"What is the purpose of this website?"* is not a self-contained question. This question can be general depending on what "this website" refers to. If this co-reference is resolved, it will be counted as a self-contained question. There are some ambiguous questions which can be self-contained and incomplete simultaneously. For example, Question 4 *"How long are therapy sessions themselves?"* can be a self-contained question if we consider therapy in general. It can also be an incomplete question because the user does not know which organization's therapy session is this question talking about. Ambiguous questions are also harder to detect but they do not affect the performance of our framework.

Table 2.10 also lists questions from the COVID-19 domain. We see questions like *"What are the symptoms in children?"* and *"When are you open for vaccines?"*. We know that these questions are asking about COVID-19 because the context is known to us. However, these sentences themselves are not self-contained. If the sentence was *"What are the symptoms in children for COVID-19?"*, it would have been a self-contained question. Finally, consider Question 9, where we see a question with a given context. In FAQs, questions with context play an important role and our module was able to extract and correctly classify these.

2.4 Related Work

The authors in [115] propose a pipeline for producing FAQ by crawling the web. Although the mentioned work addresses the same problem, the proposed approach is a semiautomated way that integrates users' feedback and usage mining to improve FAQ lists, whereas our framework is completely automated. Many works use FAQ lists/knowledge base/files for the classic task of question answering [66, 151]. The authors in [65] proposed a system where a query matches a FAQ file first and then the answer to the query is matched with one of the FAQ from that file. The authors in [34] identify missing topics in a FAQ webpage of an enterprise and suggest additional FAQ by searching the web. Although this work extracts ranked FAQ for an enterprise, it does not address the task of extracting general and self-contained question-answer pairs. This work only suggests additional questions instead of extracting every FAQ and it does not classify the question-answer pairs to be high utility (general or self-contained). Both specific and incomplete questions are extracted and suggested. The authors in [75] search the whole web to extract FAQ and then answer users' questions by retrieving the appropriate question-answer pairs. Their task is at the intersection of question answering and FAQ retrieval which is similar to [64, 114]. In their FAQ retrieval task, they rely on Google search's *"intitle:faq"* which is ineffective compared to our respective module because it misses pages which do not have "FAQ" in title. The tasks of question generation [100, 147] where questions are generated from an input passage and question answering with FAQ retrieval [78, 105, 144] where appropriate FAQ are retrieved from a knowledge base of FAQ both resemble our problem but differ in various ways. The authors in [144] argue that the number of QA pairs in a FAQ page is not enough and they leverage this issue by using a FAQ list.

We cover next the existing work that is relevant to each individual module in our framework. For Module 1, the authors in [163] propose a similar algorithm to ours; however, their algorithm deals with Twitter data only. Keyword identification from unstructured text is a common task [25,82] but while searching using retrieved keywords, they do not consider a specific type of results (e.g., FAQ) to be returned. In our work, the retrieved results are question-answer pairs. Module 2 focuses on a particular website detection but, to the best of our knowledge, most existing research has been on malicious or phishing website detection [15,95]. A work that is relevant to Module 3 is QnA Maker by Microsoft that does similar task; however, this work is proprietary. In [46], the authors extract question and answers from online forums. They address the problem of finding QAs from unstructured content. They extract every kind of QA not only high utility. In [87], the authors present a list detection algorithm to detect FAQ questions inside a webpage. The limitation of this work is that the system would require some domain knowledge to differentiate between FAQ lists and undesirable lists such as product categories. Although Module 4's task sounds like it falls under the classical problem of Question Classification [112] (i.e. classifying a question into factoid, hypothetical, etc.), it is very much different. In this work, we focus on classifying a frequently asked question into two categories: general and specific. There has been a profusion of research on text classification, from starting with bag of words to very deep convolutional networks [47, 77, 177]. Deep learning has also been used in other NLP tasks such as paraphrasing [150], slot filling [148], and intent detection [149]. Active Learning has been used in scenarios where labeled data are scarce [16, 22, 170]. We are tackling a completely new domain where the class labels for classifications are new and there is no available training data. Even though KL is a popular statistic as an input for classification problems [36], it has yet to be used as a feature.

2.5 Conclusions

We have presented a framework for extracting high-utility (i.e., general and self-contained) questions from the Web. Our framework works in a modular fashion to produce a final list of question-answer pairs. Within each module, we show that existing machine learning models are insufficient, either because they assume that large training datasets are available or because training data for each task is not available altogether. Wherever needed, we collect and annotate datasets to train our models within each module. We present extensive experimental evaluation results that show that our models within each module perform better than strong baselines, and we show that our framework indeed produces general and self-contained questions.

Chapter 3

Non-uniform History Modeling for Open Retrieval Conversational Question Answering

3.1 Introduction

Conversational Question Answering (CoQA) has recently attracted a lot of attention due to the widespread adoption of voice assistant platforms such as Siri, Alexa, and Google Assistant, and the advances in deep learning [39, 137, 174]. Given a text passage and a conversation, the goal of CoQA is to extract the answer to the last question of the conversation from the passage.

CoQA is an extension to Question Answering (QA) where the input is just one question instead of a conversation [86, 132, 159]. However, in practice users do not provide an input passage when performing QA or CoQA. This led to the newer problems of Open Retrieval QA (ORQA) [21, 45, 55] and Open Retrieval Conversational Question Answering (OrConvQA) [127], where the input is a whole document collection.

State-of-the-art works on OrConvQA (also for ORQA) employ a pipeline of three modules [58, 126, 127].

The first one is a *Retriever*, which retrieves a set of relevant passages from the collection. Both term-based (TFIDF/BM25) and embedding-based approaches may be used by the Retriever. A *Reranker* module then re-ranks the already retrieved documents to better match the question, and finally, a *Reader* module extracts an answer span from the re-ranked documents. Recent advances in transformers have produced pre-trained models like BERT which are highly effective in reader tasks [54].

A key challenge in CoQA and OrConvQA is that the final user question may have co-references or ellipses, that is, some terms may refer to terms in the past conversation, while other useful contexts may be missing from the question. Further, previous (historic) turns of the conversation may add valuable context to the question being asked. Clearly, some of the past turns may be more useful than others as context for the last question. Blindly adding all turns may lead to a noisy history model. Including all turns may also be infeasible for some models like BERT, which can only support 512 tokens as the query and passage.

Previous CoQA works propose different approaches to model the conversational history: some append all history turns to the final query making it a one big query and use it to retrieve the answer [39, 137], or use a backtracking algorithm which selects/disregards

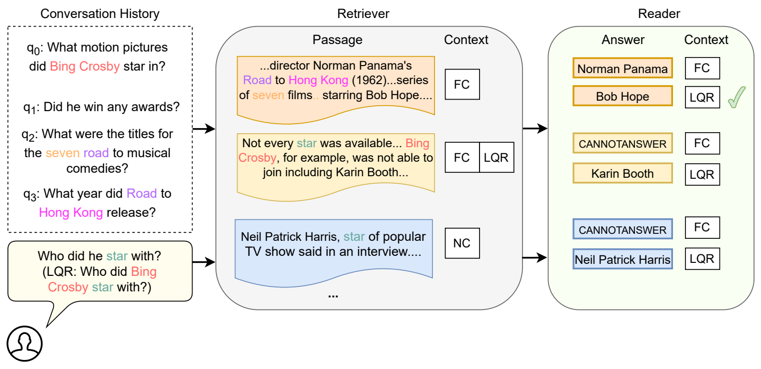

Conversation History
q_0: What motion pictures did Bing Crosby star in?
q_1: Did he win any awards?
q_2: What were the titles for the seven road to musical comedies?
q_3: What year did Road to Hong Kong release?
Who did he star with? (LQR: Who did Bing Crosby star with?)
Retriever
Passage
Context
...director Norman Panama's Road to Hong Kong (1962)...series of seven films starring Bob Hope....
FC
Not every star was available... Bing Crosby, for example, was not able to join including Karin Booth...
FC
LQR
Neil Patrick Harris, star of popular TV show said in an interview....
NC
...
Reader
Answer
Context
Norman Panama
FC
Bob Hope
LQR
CANNOTANSWER
FC
Karin Booth
LQR
CANNOTANSWER
FC
Neil Patrick Harris
LQR

Figure 3.1: Example of the impact of non-uniform conversational history modeling. Full Conversational Context (FC) retrieves the most relevant passages in the *Retriever* module, while a narrower context, Last Question Rewrite (LQR) predicts the correct answer span in the *Reader* module.

a particular history turn using deep reinforcement learning [125], or rewrite the final query using the context of the whole conversation [109, 152, 158].

Previous OrConvQA works either use the previous 6 turns [126, 127] or all turns with predicted answers [58] as context.

Despite the different history modeling approaches of these previous works, they all use the same history model for all three modules of the pipeline. We hypothesize that this is suboptimal. Specifically, our hypothesis is that as we move towards the right of the *Retriever*→*Reranker*→*Reader* pipeline and the number of the input passages (or documents) decreases, the history context should become shorter and more focused. That is, the Retriever should have access to broader context to not miss any relevant documents, whereas the Reader should have little context to help it identify the exact text span that answers the user question.

For example, in Figure 3.1 we see that modeling the history with Full Conversational Context (FC) returns the most relevant passage (top one) which has all the necessary information needed to answer the query such as movie name *Hong Kong*, director's name *Norman Panama*, and co-star's name *Bob Hope*. In contrast, narrower context – Last Question Rewrite (LQR) or No Context (NC) – returns suboptimal passages that do not contain the answer. Specifically, LQR returns a passage related to the topic of the conversation *(Bing Crosby)* but does not contain movie information which is an important context found in the history, and NC returns a passage that does not contain any relevant information. Once documents are retrieved, additional context (FC) may act as noise for the *Reader* module, whereas more focused context (LQR) is able to extract the right span. FC returns the wrong answer (*Norman Panama*) as it gets confused and associates movie name from the context *Road to Hong Kong* to a person's name *Norman Panama*. As LQR focuses on *starring* and *Bing Crosby*, it identifies the correct answer.

In addition to using the same context, the state-of-the-art pipelines [58, 126, 127] and history modeling approaches [109,125,158] depend fully on training data to fine-tune the *Retriever*, *Reranker*, and *Reader* modules. Finding quality training data for various domains of OrConvQA datasets is challenging. Ideally, a pipeline should be domain-agnostic and use appropriate history modeling to capture the context. In this chapter, we contribute in three ways towards solving the OrConvQA problems: (a) we propose NORMY[1], the first unsupervised solution pipeline, (b) we build and publish a new dataset, and (c) we implement and experimentally compare various state-of-the-art history modeling algorithms for each of the three modules of the *Retriever*→*Reranker*→*Reader* pipeline.

[1]**N**on-Unif**ORM** Histor**Y** Modeling

Table 3.1: Comparison of selected tasks and datasets on the dimensions of Question Answering(QA), Open Retrieval(OR), Conversational (Conv), History Modeling (HM) and Non-uniform History Modeling (NHM)

Task/Dataset	QA	OR	Conv	HM	NHM
NQ [86], SQuAD [132]	✓	✗	✗	✗	✗
TriviaQA [76],					
MSMarco [21],DrQA [35]	✓	✓	✗	✗	✗
CoQA [137],					
QuAC [39],ShARC [143]	✓	✗	✓	✗	✗
HAE [128], RL [125], RW [158]	✓	✗	✓	✓	✗
OrConvQA [127], d2d [60]	✓	✓	✓	✗	✗
NORMY**[ours]**	✓	✓	✓	✓	✓

Our proposed system, NORMY, uses a *non-uniform* history context for the three pipeline modules.

We also propose a novel history modeling algorithm for the *Retriever* module that produces improved results over state-of-the-art baselines. Unlike previous approaches where the passages retrieved in previous turns are discarded, our Retriever algorithm considers past passages as candidates and proposes a ranking function that combines turn-based decay with context-based reranking of each passage. This ensures we do not miss an important passage due to noise being added in later turns. Table 3.1 summarizes the related work landscape, where we see that none of the previous work addresses all aspects of the problem. Only NORMY is question answering, open retrieval, conversational, performs history modeling.

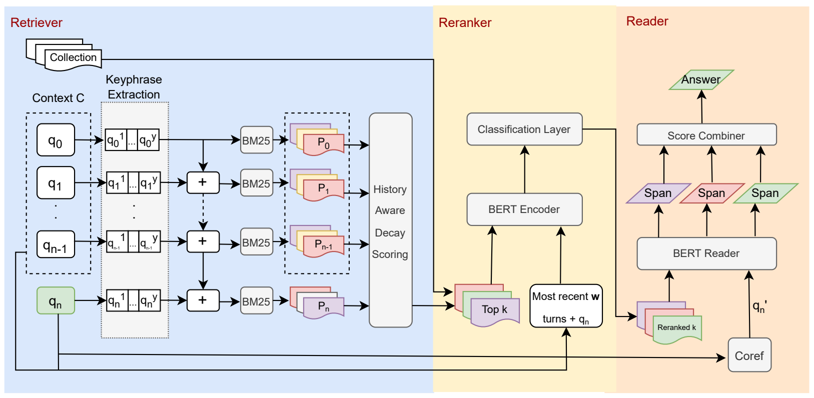


Figure 3.2: The architecture of NORMY. The input is the current question q_n, all history questions q_i^{n-1}, and the document collection D. The *Retriever* module models the history using keyphrase extraction per history turn and retrieves passages $P_0 \cdots P_k$ using BM25. Our novel History Aware Decay Scoring module refines all returned passages and outputs top-k. The *Reranker* reranks the passages using most recent w turns and *Reader* uses coreference resolution to rewrite the last query q_n and outputs the best answer span combining all three modules' scores.

We evaluate our individual modules and the overall pipeline using three varied datasets. First, we use the *ORQUAC* dataset [127], which is an extension of the CoQA dataset [137]. A drawback of this dataset is that it does not portray natural dialogue conversation, as the chat is limited to asking questions and getting answers. Thus, we selected the *doc2dial* [60] dataset for additional evaluation. However, this dataset is not created for the open retrieval conversational QA task as there are only a small number of documents as a corpus and the focus was to generate natural language answers and not text spans. For that, we created an updated *doc2dial* dataset, which we call *doc2dial-Or*. Third, we conduct experiments on ConvMix [40], where the corpus includes single-sentence passages and the history turns contain fewer co-references than previous datasets mentioned. NORMY outperforms the state-of-the-art in all three datasets.

In summary, we make the following contributions in this chapter:

- We identify the problem of uniform history modeling in conversational QA and propose the first end-to-end pipeline for OrConvQA that uses non-uniform history modeling.
- We propose NORMY, a new unsupervised non-uniform universal history modeling pipeline. NORMY employs a novel history modeling approach for the Retriever module, which builds on keyphrase extraction principles, and leverages returned passages from previous history turns.
- We perform an extensive comparison and analysis of various history modeling techniques for each module of the pipeline, on three diverse and structurally different datasets, and show that using the same modeling is suboptimal.
- We expand the *doc2dial* dataset for the OrConvQA task and make our full source code and dataset available to the community [2].

3.2 Problem Definition and Overview of NORMY

Problem Definition. The input to the OrConvQA problem is a question q_n, the conversational history $C = q_0, \cdots, q_{n-1}$, and a document collection D. As in previous work, the history does not contain the answers to the questions, we also assume no access to the ground truth answers [127]. The output is an answer span a_n, extracted from one of the documents in D, which best answers q_n. The solution pipeline is shown in Figure 3.2.

[2]https://github.com/shihabrashid-ucr/normy

There are two key decisions we have to make for each module. First, pick what algorithm to employ (e.g. BM25 [140] or BERT [54] and so on) and with what parameters. Second, define what conversational context C to input to the algorithm. In this work, we employ the state-of-the-art algorithm for each module and focus on the choice of conversational context for each module.

Overview of NORMY. NORMY, as shown in Figure 3.2, generates a different model of the conversational history for each module of the pipeline. Given the Retriever's history model discussed below and the collection, the *Retriever* selects the top k passages using BM25. Then, using a history of the last w turns, the *Reranker* reranks the k passages using transformer-based similarity measures. Finally, the transformer-based *Reader* module models the history by rewriting the final query into a self-contained query, using coreference resolution, to find the best answer span. The answer span with the highest combined score from all three modules is the final answer. Note that our whole system is designed in an unsupervised fashion. There is no training data needed.

3.3 Modules of NORMY

3.3.1 Retriever

The *Retriever* module retrieves the k most relevant passages from a document collection D, given a query q_n and context C. We considered two types of search algorithms: classic Information Retrieval BM25-style ranking methods, and dense retriever methods. Although dense retriever approaches like ORQA [91] and DPR [79] which use encodings of documents using ALBERT [88], have shown to provide better results, they require training data for fine-tuning. Their vanilla pre-trained models without training do not perform as well as BM25. Further, BM25 is more scalable for large collections. Hence, given that the main focus of this chapter is history modeling, we picked BM25 for our Retriever. Specifically, we index the documents using Lucene[3]. Then we retrieve top k documents using BM25, which is a term frequency based document ranking algorithm.

History Modeling. NORMY's Retriever has two key novelties. First, we use a *keyphrase extraction*-based candidate selection algorithm to identify the key context from the whole conversational history. Second, we consider all passages returned by previous turns alongside passages returned by final turn as candidate passages, and rank them using *history aware decay scoring* method to return top k.

Our retrieval algorithm is shown in Algorithm 2. We extend the keyphrase extraction algorithm YAKE [32] to select y best keywords per conversation turn using the YAKE formula shown in Equation (3.1). YAKE considers features like the casing of the word, word positions, word frequencies, word relatedness to context, etc. to assign a score $S(b)$ to each word b.

$$S(b) = \frac{W_{Rel} \cdot W_{Pos}}{W_{Case} + (W_{freq}/W_{Rel}) + (W_{DifS}/W_{Rel})} \tag{3.1}$$

where W_{Rel} is the relatedness to context score, W_{Pos} is the word position score, W_{Case} is the word casing score, W_{freq} is the word frequency divided by the sum of mean term frequency and standard deviation σ, and W_{DifS} is calculated based on how many times a particular word appears in other sentences. The detailed equations of all the terms can be found in [32]. We compute the union $R(C)$ of the reformulated questions $R(q_0) \cdots R(q_n)$ to

[3]https://lucene.apache.org/pylucene/

Algorithm 2 NORMYRetriever

```
Require: Context $C, q_n$
Ensure: SEL: Top $k$ returned passages
1: $SEL \leftarrow []$, $P \leftarrow []$
2: for each turn $i \in 1 \cdots n$ do
3:     $R(q_i) \leftarrow YAKE(q_i)$
4:     $R(C) \leftarrow R(q_0) \cup \cdots \cup R(q_i)$
5:     $P_i \leftarrow Retrieve_k(R(C))$ //top-k by BM25
6:     $P \leftarrow P \cup P_i$
7:     for each passage $p \in P_i$ do
8:         Compute score $S_{rt}(p)$ using Eq. 2
9: $SEL \leftarrow$ SelectTopk($P$) //based on $S_{rt}(p)$
10: return $SEL$
```

retrieve the top k passages (line 4-5). Each passage returned has a BM25 score assigned to it.

History-Aware Decay Scoring. After retrieving k passages for the current turn n, we refine their scores by considering their similarity to the retrieved passages. Further, we assign less weight to passages returned from previous turns. Specifically, we use a decay weight λ to update the scores of older passages. The passages returned from previous turns may be relevant for subsequent modules but they do not share equal weight to passages returned from current turn n. Next, to assess the relevance of each passage P_{nj}, $\{j = 1 \cdots k\}$ from turn n, we compute the average pairwise similarity with passages returned in the previous turn $P_{(n-1)i}$, $\{i = 1 \cdots k\}$. This ensures that the passages returned has relevance to the whole conversation. Passages returned from irrelevant conversation turns will be scored less. To compute the similarity, we use SBERT [138] to produce embeddings of passages and perform cosine similarity. We update the score of P_{nj} using this similarity. Finally, we rank all the passages using updated scores S_{rt} and select top k. The retriever score of a passage p is shown in Equation (3.2).

$$S_{rt}(q_n, C, p) = max(BM(R(C \cup q_n), p) - \lambda, 0) \cdot \sum_{i=1}^{k} sim(p, P_{(n-1)i})/k \tag{3.2}$$

where *BM(.)* is the BM25 score of a passage and *sim(.)* returns semantic similarity between two passages.

3.3.2 Reranker

The *Reranker* module reranks the retrieved top k passages using transformer-based encoders and a neural network to compute passage relevance score. The transformer based Reranker augments BM25 ensuring an extra layer of passage relevance. As k <<*total size of collection*, using a transformer encoder is inexpensive.

A Reranker has been shown to improve the overall performance of the end-to-end system with little additional cost [68, 127]. However, we show that using the same history modeling as the previous module or using the context from all history turns do not give the best results as now we have grounded documents as evidence. Our experimental results show that using a context with a history window size w works best. The input to the module is the final query q_n, the context C, and k passages retrieved by the Retriever. A reranking score S_{rr} is assigned to every passage. **Encoder.** Our Reranker module uses BERT to encode the input representation. We use the last w history turns before q_n and concatenate them together to model the history. We then concatenate the retrieved passage p_j, $j = \{1 \cdots k\}$ to the appended history turns to create the final input sequence (q_n, C, p_j) = [CLS] q_{n-w} [SEP] $\cdots$ [SEP]q_{n-1} [SEP]q_n [SEP]p_j. We use the contextualized vector representation of the input sequence $\nu_{[CLS]}$, and use it as input to a fully connected feed-forward layer that classifies the given passage as either *relevant* or *non-relevant* and outputs a classification score S_{rr}:

$$\nu_{[CLS]} = W_{[CLS]} BERT(q_n, C, p_j)_{[CLS]} \tag{3.3}$$

$$S_{rr} = P(Rel = 1 | q_n, C, p_j) \overset{\triangle}{=} softmax(\nu_{[CLS]}) \tag{3.4}$$

where $\nu_{[CLS]} \in \mathbb{R}^T$, T is the model embedding dimension, which is 768, and $W_{[CLS]}$ is a projection of the $[CLS]$ representation to obtain the sequence representation $\nu_{[CLS]}$. We compute the score for each passage in top k independently and rerank them based on S_{rr}.

3.3.3 Reader

The *Reader* module inputs the final query q_n, the context C and the reranked passages $\{p_1, p_2 .... p_k\}$ and outputs a span from one of the passages as the answer.

History Modeling. As the documents have already been narrowed down using the conversational context in previous modules, we show that using a history modeling with full contextual information produces worse results than a history model that uses less context. This happens due to: 1) The passages already hold the necessary contextual information from the history, 2) Previous history questions in the context misdirects the BERT Reader model into predicting incorrect answer spans. The naive idea would be to use just the final query as input. However, the final query is prone to co-references and ellipses as users will not use self-contained utterances in a natural conversation. Thus, we use a co-reference resolution model to generate a resolved final query ${q_n}'$ using the previous context. We adapt the huggingface neural co-reference model [4] which uses two neural networks to assign a score to each pair of mentions (or co-references) in the input and their antecedents [44]. The history turns q_0 to q_{n-1} are concatenated and used to rewrite q_n into ${q_n}'$.

Encoder. Our Reader module uses similar BERT architecture as the previous module to encode the input. The input sequence "$[CLS]{q_n}'[SEP]p_j$" is used to generate a representation of all tokens in the input. Two sets of parameters, a start vector W_s and an end vector W_e are used to compute the score for the m-th token.

$$\nu_{[m]} = BERT(({q_n}', p_j))_{[m]} \tag{3.5}$$

$$S_s({q_n}', p_j, [m]) = W_s \nu_{[m]} \quad S_e({q_n}', p_j, [m]) = W_e \nu_{[m]} \tag{3.6}$$

where S_s is the start score of a token and S_e is the end score. The span Reader score S_{rd} is computed as the maximum score of each token being either the start or end token. The start token must appear before the end token in the input.

$$S_{rd}({q_n}', p_j, s) = \underset{[m_s],[m_e] \in ({q_n}', p_j)}{max} S_s({q_n}', p_j, [m_s]) + S_e({q_n}', p_j, [m_e]) \tag{3.7}$$

where s is the answer span with the start token $[m_s]$ and end token $[m_e]$. The answer spans are re-ranked using the combined score of all three modules and the top answer is given as a prediction.

$$S(q_n, C, p_j, s) = S_{rt}(q_n, C, p_j) + S_{rr}(q_n, C, p_j) + S_{rd}({q_n}', p_j, s) \tag{3.8}$$

[4]https://huggingface.co/coref/

Table 3.2: Dataset Statistics

	ORQUAC	doc2dial-OR	ConvMix
# Dialogues	771	661	1679
# Questions	5571	4253	2284
# Avg tokens/qstn	6.7	10	6.39
# Avg tokens/ans	12.2	21.6	2.17
# Avg questions/conv	7.2	6.4	5.00
# Passages	11M	11.6M	5.94M

3.4 Experimental Evaluation

3.4.1 Datasets

We use three datasets with different conversation structures. The first dataset: ORQUAC [127] is an aggregation of three existing datasets: QuAC, CANARD [56], and Wikipedia corpus that serves as a knowledge source for open retrieval. The Wikipedia corpus is a collection of 11 million passages that are created from splitting Wikipedia articles into 384 tokens. The second dataset is *doc2dial-OR*, which was created by us, as an extension of *doc2dial* [60] dataset, which consists of natural information-seeking goal-oriented dialogues that are grounded in documents. doc2dial has more complex questions than ORQUAC, associated with multiple sections of a document. However, this dataset is only grounded to 480 long documents collected from different government websites, which is not ideal for an open retrieval task. *doc2dial-OR* extends doc2dial by having a much larger set of passages consisting of (a) 11 million Wikipedia passages[5], and (b) the 480 documents of doc2dial

[5]https://dumps.wikimedia.org/enwiki/20191020

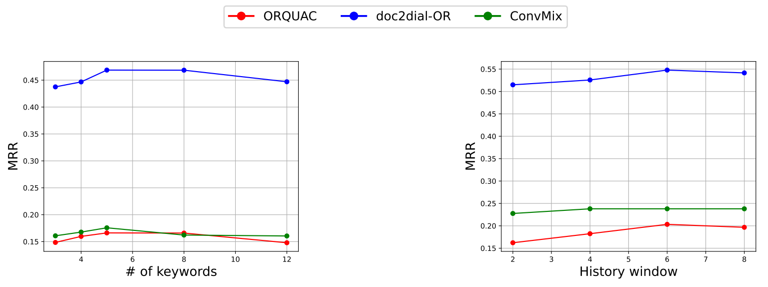
Table 3.3: Retriever Results

Setting	ORQUAC				doc2dial-OR				ConvMix			
	MRR	R@1	R@5	R@10	MRR	R@1	R@5	R@10	MRR	R@1	R@5	R@10
No History	0.0312	0.0177	0.0516	0.0649	0.2564	0.2003	0.3320	0.3907	0.1027	0.0696	0.1449	0.186
First Last	0.1174	0.0739	0.1757	0.2249	0.4283	0.3378	0.5523	0.6381	0.1586	0.109	0.2263	0.281
Full History	0.1361	0.0785	0.1748	0.2178	0.4289	0.3376	0.5584	0.6433	0.1605	0.1077	0.2364	0.2911
Fixed window	0.1222	0.0827	0.1744	0.2189	0.4292	0.3378	0.5584	0.6433	0.1605	0.1077	0.2364	0.2911
Backtracking	0.1160	0.0726	0.1742	0.2238	0.4226	0.3301	0.5494	0.6364	0.1696	0.1169	0.2429	0.3104
Rewriting	0.0516	0.0307	0.0814	0.1080	0.2605	0.2031	0.3369	0.3985	0.12	0.0827	0.1654	0.2123
NORMYRet	**0.1662**	**0.1147**	**0.2367**	**0.2891**	**0.4687**	**0.3809**	**0.5906**	**0.6780**	**0.1757**	**0.119**	**0.2513**	**0.3139**

split into 384-token chunks. The second difference between doc2dial-OR from doc2dial is that we convert free-text ground truth answers to exact text spans from the gold passage in the dataset, to make it suitable for a span prediction task, as is the case for ORQUAC. The third dataset is ConvMix [40], which contains documents from heterogeneous sources: Wikipedia info boxes, tables, and text snippets (passages). They use the Wikipedia dump from 2022-01-31. To adapt this dataset for our task, we selected the 5.94 million textual snippets as our collection and the question turns in a conversation where the answer can be extracted from these text snippets. The dataset statistics are shown in Table 3.2.

3.4.2 Experimental Setup

Competing History Models. To the best of our knowledge, there are no fully unsupervised non-uniform history modeling approaches. There are supervised systems (OrConvQA [127], WS-OrConvQA [126]) that uses history window of 6 for all the modules. ConvADR-QA [58] uses all history turns along with their predicted answers as context. However, they require annotated data to train the modules. We adapt their approaches to



(a) Retriever performance with y (b) Reranker performance with w

Figure 3.3: (a) and (b) subgraphs show the impact of number of keywords **y** and history window size **w** for Retriever and Rearanker modules respectively.

an unsupervised setting. There are also conversational closed retrieval systems (RL [125], RW [158]). We adapt such history modeling methods to an open-retrieval setting. Further, we also propose some standard history modeling techniques. Thus, we have identified the following baselines:

1) No History [39]: Where we do not perform any history modeling. The last question turn is used as input to each individual module.

2) First-Last: We propose an intuitive history modeling baseline, where we define the history as the combination of the immediately previous user utterance and the first utterance of the conversation.

3) Full History: With all previous turns concatenated. For the modules where we use transformer models with token limitation, we prune the earlier tokens if the total token size exceeds 384. The input sequence is $C = [CLS]q_0[SEP]q_1\ [SEP]\cdots[SEP]q_n$

4) YAKE [32]: Keyphrase extraction-based history modeling has not been used previously in any research work. We extract y keyphrases per history turn and concatenate them with the last turn to create the input sequence. The keyphrases are extracted using a keyword extractor tool built with Python.

5) Backtracking [RL]: We adapt the *immediate reward* based history selection proposed by Qiu et al. [125] for closed retrieval systems. We select a history turn if the similarity with previously selected history turns is greater than 0.5. For each history turn($i = 1...n$) we calculate the immediate reward with SBERT sentence encoder.

6) Question Rewriting [RW]: We adapt the algorithm of Vakulenko et al. [158] for closed retrieval systems, which uses a question rewriting model to resolve ambiguous questions (co-references) into self-contained questions. Their model requires training data thus we use *neuralcoref* to resolve the co-references of the final query using previous history turns. There are other query rewriting models like QReCC [17] which also require annotated data.

7) Fixed Window [OrConvQA, WS-OrConvQA] [126,127]: WS-OrConvQA is an improvement over OrConvQA but uses training data to learn weak supervision signals. Both models use a history window size w. Window size 6 is shown to produce the best results for both. To select the baseline, we also compared different window sizes (2,4,6,8) in a small validation set and $w = 6$ has given the best results.

8) Fixed Window with Ans. [ConvADR-QA] [58]: This model predicts the answers for each historical question with a teacher model using annotated query rewrites and appends the predicted answer to the context along with historical questions. For a fully unsupervised pipeline, we adapt this approach to predict an answer for each question using OrConvQA and append the answer to the context. The input sequence is $C =$

Table 3.4: Reranker Results

Setting	ORQUAC				doc2dial-OR				ConvMix			
	MRR	R@1	R@5	R@10	MRR	R@1	R@5	R@10	MRR	R@1	R@5	R@10
No History	0.1408	0.1033	0.1927	0.2891	0.4572	0.3592	0.5901	0.6780	0.2264	0.1834	0.285	0.3139
First Last	0.1779	0.1491	0.2179	0.2891	0.5247	0.4514	0.6301	0.6780	0.2396	0.2052	0.2955	0.3139
Full History	0.1996	0.1702	0.2402	0.2891	0.5401	0.4644	0.6415	0.6780	0.2405	0.2020	0.2944	0.3139
NORMY	**0.2033**	**0.1767**	**0.2411**	**0.2891**	**0.5478**	**0.4747**	**0.6515**	**0.6780**	0.2405	0.2020	0.2944	0.3139
Backtracking	0.1954	0.1661	0.2361	0.2891	0.5325	0.4635	0.6368	0.6780	0.2403	0.2017	0.2940	0.3139
Rewriting	0.1696	0.1344	0.2181	0.2891	0.4583	0.3604	0.5911	0.6780	0.2295	0.1873	0.2867	0.3139
YAKE	0.2008	0.1731	0.2387	0.2891	0.5396	0.4624	0.6377	0.6780	**0.2418**	**0.2031**	**0.2946**	**0.3139**

$[CLS]q_0a_0[SEP]q_1a_1\ [SEP]\cdots[SEP]q_na_n$, where a_n is the predicted answer for turn n. We use the pipeline to predict the answer first and then append.

Implementation Details. NORMY is fully unsupervised without requiring any training data. The pre-trained models are implemented with the open-source library Huggingface [6]. We index our document collection using PyLucene with *StandardAnalyzer* as tokenizer Indexing is done with term frequencies, document frequencies, and positions. Keyphrase extraction is done with open source library *pke* [28]. For computing similarity, we use SBERT's pre-trained *roberta-large* model. For *Reranker* module, we use pre-trained BERT model finetuned on MSMARCO dataset [116]. For our *Reader* module we use a pre-trained *bert-large* model finetuned on SQUAD dataset. We make our full code available to the research community.

3.4.3 Retriever Results

Evaluation Methodology. We use the commonly used *Mean Reciprocal Rank (MRR)* and *Recall (R@k)* methods to measure our retrieval performance. *MRR* calculates

[6]https://github.com/huggingface/transformers

how far down the ranking the first relevant document is on average, where higher is better. $R@k$ measures the fraction of times the correct document is found in the top k predictions, where higher is better.

Results. We first experiment with different values of the number of keywords y in the keyphrase extraction shown in Figure 3.3a and find that $y = 5$ performs best. We use $y = 5$, $k = 10$ and $\lambda = 0.1$ in Table 3.3. We found that using a larger k increases the execution time of the system with a very small accuracy benefit. Figure 3.3 shows our experimental results for different parameters used in NORMY.

We compare different history modeling techniques using BM25 in Table 3.3. For all three datasets, we see that our NORMY Retriever performs significantly better than other baselines. This is due to a couple of factors: 1) We remove irrelevant information from each history turn rather than eliminating the history turn altogether, 2) We consider previously retrieved passages from previous history turns as candidate passages. We can also see that history models that use fewer contexts like Question Rewriting, First-Last, and No History perform significantly worse indicating we need more context while retrieving from millions of passages.

3.4.4 Reranker Results

Evaluation Methodology. We use the same metrics as the Retriever as both of these modules produce top k passages.

Results. From Table 3.4 we see that the Reranker module significantly improves the ranks of relevant passages. Using a fixed window, which sits between a broader context like Full History and a narrower context like Rewriting produces the overall best results.

We conduct experiments with different history window sizes w and show the results in Figure 3.3b. Fixed window of 6 works best for two of our datasets, which is supported by the literature [127]. For ConvMix, we see that YAKE performs slightly better than all other methods and the results vary very little. This is because the passages in the collection are single sentences only, leading to multiple passages being relevant to the question. The *Reranker* ranks such passages similarly whereas only one contains the gold answer. In the real world, it is unusual for the documents to be single sentences. Note that our hypothesis that the *Reranker* needs less context than the *Retriever* still holds, as YAKE has less context than Full History.

Table 3.5: Reader F1

Setting	ORQUAC	doc2dial-OR	ConvMix
No History	0.1557	0.1898	0.6785
First Last	0.0996	0.1291	0.4842
Full History	0.0845	0.1196	0.3711
Fixed Window	0.0848	0.1200	0.4859
Backtracking	0.1118	0.1541	0.5591
NORMY(Rewriting)	**0.1774**	**0.2220**	**0.7393**
YAKE	0.1277	0.1551	0.67

3.4.5 Reader Results

Evaluation Methodology. We treat the evaluation of Reader module as a span selection task and adopt token level F1 as the evaluation metric. F1 calculates the similarity between the ground answer and the predicted span, where higher means better.

Table 3.6: Entire Pipeline F1. ‡ means statistically significant improvement over baseline with $p < 0.5$.

Setting	ORQUAC	doc2dial-OR	ConvMix
NORMY[ours]	**0.0782**‡	**0.1625**‡	**0.1723**‡
NORMY w/o decay	0.0668‡	0.1323‡	0.1490
NORMY w/o sim	0.0695‡	0.1431‡	0.1562‡
OrConvQA [127], WS-OrConvQA [126](BM25)	0.0478	0.0955	0.1314
OrConvQA, WS-OrConvQA (DPR)	0.0466	0.0948	0.1298
ConvADR-QA [58]	0.0454	0.0897	0.1244

Results. From Table 3.5 we can see significant performance drops when more contexts are added for all three datasets.

As the candidate passages have been reduced to 1, transformer-based reader model performs significantly better when only one query is used.

Narrower contexts like adding no history and question rewriting perform much better than broader context models further proving our hypothesis. Among them, question rewriting produces a better result. For ConvMix we can see very high F1 scores for appropriate history models as the answers are on average two tokens only, which makes the *Reader* model easily predict answer spans. However, history models with broader context have poor F1 scores for this same dataset indicating the model needs proper history models even in simpler scenarios.

3.4.6 End-to-end Evaluation

In this section, we compare our pipeline with the state-of-the-art models [58, 126, 127], which either use a fixed window of 6 or full history with predicted answers uniformly in all modules. We see in Table 3.6 that SOTA models perform poorly in a fully unsupervised setting. We also see that, ConvADR-QA performs worse than OrConvQA, as in an unsupervised setting, a wrongly predicted answer could misdirect the context to retrieve irrelevant passages. Our pipeline with non-uniform history modeling performs significantly better. SOTA models use fine-tuned dense retriever model (DPR) for their retriever module instead of BM25 which is used by NORMY. We use the vanilla pre-trained version of DPR here as we don't have access to training data. We also compare to a variant of OrConvQA that uses BM25 instead of DPR for completeness. We see that BM25 performs better than dense retriever models. Note that, uniformly using other baseline history modeling techniques evaluated in previous subsections does not produce better results than NORMY in the end-to-end pipeline. We do not show these in Table 3.6 for brevity.

3.4.7 Ablation Studies

The effectiveness of our model relies on some of the design choices we made. We investigate such choices our novel retriever *NORMYRetr* has from equation (3.2). We present the ablation results in Table 3.6. Specifically, we show two ablation settings as follows:

NORMY w/o decay. We showed that if we disregard previously returned passages we may miss out on some relevant information required for subsequent modules. However, if we gave the same weight to previous passages as passages returned from current turn n, we see a degradation in performance. This is due to the current turn holding the most amount of information. Thus passages returned from the current turn should be given the most weight.

NORMY w/o sim. If a history turn is related to its previous turn, the passages returned will also have some similarities. Here, we disregarded the average pairwise similarity with the previous turn's returned passages from equation (3.2). We again see a decrease in model performance. By refining retriever scores with similarity score, we compute the relevance of passages with relevant conversational history.

The ablation studies further verify that both decay and similarity scores are crucial for NORMY to perform best.

3.5 Related Work

Machine Reading Comprehension (MRC). MRC task typically includes a single-turn query where the answer grounds in a short passage. It started with TREC [159] in the early days where the goal was to retrieve the appropriate passage for 200 factoid questions and advanced to recent high-quality datasets like NQ [86], SQuAD [131, 132], NewsQA [157].

Open Domain QA. Open domain QA introduces large corpus as grounded documents and the task is to retrieve the appropriate documents and then try to extract the answer span. For this task, high-quality datasets have been proposed such as TriviaQA [76], MSMarco [21], Quasar [55], WikiQA [45], PATQuestions [107]. Some previous work [84, 90, 164] selects answers from a closed set of passages or learns to rerank them. End-to-end open domain pipelines like DrQA [35] and BERTserini [171] use TFIDF/BM25 for the retrieval of passages and a neural reader to select the answer span. ORQA [91] and

DPR [79] introduce a learnable retriever module with a dual encoder architecture. They show scalability for large scale collections [53]. These works are all single-turn QA's whereas we target multi-turn conversations.

Conversational QA. Conversational QA is a variant of MRC where the queries are no longer single turn and the role of retrieval is disregarded [41]. The multi turn questions can be interconnected (CoQA [137], DoQA [31]), can depend on the previous history answer(QuAC [39]) or only limited to binary answers (ShARC [143]). A better understanding of the context of conversation history is needed to answer the grounded question. To capture the context, FlowQA [69] and GraphFlow [38] use each word as nodes in a graph and use an attention mechanism to represent the history; HAE [128] considers the history ground answers as context which is impractical for real life dialogue agents; Pos-HAE [129] considers the history turn positions as additional encoding. There are also backtracking based [125] and query rewriting based [17, 109, 158, 160] models as mentioned in previous sections.

Open Domain Conversational QA. ODQA models like OrConvQA [127], WS-OrConvQA [126], ConvADR-QA [58] do not perform any history modeling and use the same history window in all of their modules for the end-to-end system. Other ODQA works like TopiOCQA [13], graph-OrConvQA [94] do not focus on history modeling and use gold training data to train neural models for their pipeline. Similar to *Extractive QA* like this task (OrConvQA), there are *Abstractive* pipelines [83, 93] where the answer is generated using transformers like BART [92] and T5 [130] rather than being extracted from passages. Such pipelines are sequence-to-sequence tasks and not span predictions. There are other works that perform ODQA over structured data such as knowledge graphs [41] or a combination of data sources [40]. They do not perform any history modeling to contextualize the conversation and the task is out of the scope of this work.

3.6 Conclusion

We have presented the first end-to-end pipeline that uses non-uniform history modeling for open retrieval conversational question answering. We show that existing systems are suboptimal due to modeling the context in the same way for all modules and not utilizing previously returned passages for the *Retriever* module. We have also proposed a novel algorithm to utilize such passages to output higher-quality passages for subsequent modules. We further updated the doc2dial dataset to make it appropriate for OrConvQA task. Extensive experimental evaluation from various history modeling techniques with different types of data shows that NORMY significantly outperforms the state-of-the-art in each individual module and the entire pipeline.

Chapter 4

Budget-constrained Text Re-ranking with Large Language Models

4.1 Introduction

Text re-ranking focuses on ranking N source documents given a specific query and is crucial for providing the relevant retrieved context to downstream tasks. It serves either as a standalone task or as an intermediate step for question answering tasks [134, 136] in a retrieval augmented setting, where the answer is generated from the top k relevant passages. Traditional ranking methods includes BM25 [97] and neural methods like DPR [79], Contriever [71] etc. Recently, large language models (LLMs) such as GPT-4 [120] have demonstrated dominant performance in text re-ranking [154].

However, utilizing LLMs often comes at a cost: the process can be quite expensive, as closed-source LLMs charge based on the number of tokens. This expense escalates in text re-ranking due to the need to input substantial text, proportional to the number of passages to re-rank. For example, re-ranking 500 passages for a single query, with each passage having an average length of 100 tokens, currently costs at least 5 USD when using GPT-4 [4]. This cost becomes intractable when businesses need to handle thousands of queries daily, making *budget* the biggest constraint.

Although there are many alternatives such as TextSynth [7], AI21 [1], Cohere [2], Replicate [6], etc., that offer LLM API services at lower costs, utilizing these commercial APIs may still not be sustainable with high volumes of queries. This motivates us to investigate the trade-off between cost and performance for text re-ranking. Our aim is to propose a budget-aware solution for text re-ranking that maximizes performance within the constraints of a given budget. With this goal in mind, we approach budget-constrained text re-ranking using LLMs as a constrained optimization problem. Here, we explore various re-ranking methods with different properties and optimize for the best strategy for budget-aware text re-ranking.

Our work contributes to the first efforts in budget-aware modeling utilizing LLMs for text re-ranking, to the best of our knowledge. Recent work on cost-aware applications of LLMs with *LLM Cascading* [37, 145, 173] is not applicable to text re-ranking. They either focus primarily on QA or reasoning tasks [173], or require a fine-tuned model (with training data) to assess the generation quality of LLMs [37, 145]. On the other hand, works focusing on using LLMs for text re-ranking primarily aim at performance improvement without

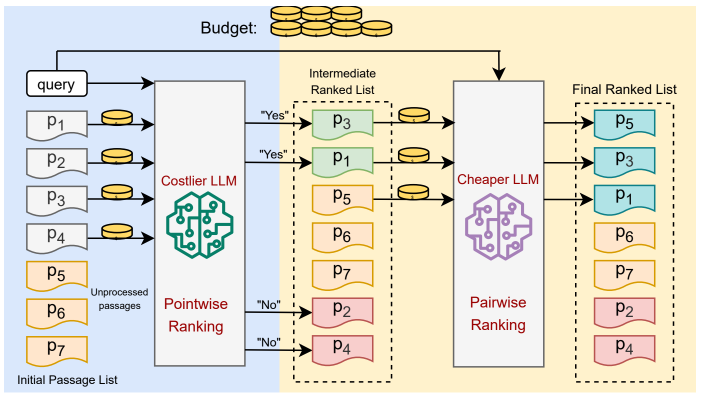


Figure 4.1: An overview of EcoRank with an example of 7 passages. A fraction of budget is spent on 4 passages for pointwise prompt with a costlier LLM and an intermediate ranked list is generated with unprocessed passages in the middle. Then, using the rest of the budget we call the cheaper LLM to do pairwise comparisons and create the final ranked list.

considering budgets. The three most common approaches exhibit increasing costs as the number of tokens inputted into the LLMs increases: 1) Pointwise prompts [142], which input a single passage per request and output calibrated prediction probabilities before sorting; 2) Listwise prompts [102, 154], which input multiple passages per request as lists and ask LLMs to output the lists in order; 3) Pairwise prompts [124], which input pairs of passages per query per request and use a sliding window to sort the top-k passages. Approximately, the most expensive approach, namely pairwise prompts, can cost about $2 \cdot k$ times more than pointwise prompts where k is the size of the sliding window.

In this chapter, we propose a suite of budget-constrained methods to perform text reranking using a set of LLM APIs. Our most efficient method, which we refer as EcoRank, is a budget-constrained LLM-based text re-ranking pipeline, that jointly optimizes several objectives: 1) which prompt designs to deploy, 2) which LLM APIs to call, and 3) how to split budget between multiple prompts and LLMs. Optimizing all these decisions jointly is challenging for the following reasons. (a) The provided budget may not be enough to input all input texts once to the LLM API. The developed method must be able to optimize for the top-few (e.g. top-1) text. (b) Different LLM APIs may have different strengths and limitations. (c) There is an exponential number of combinations of prompt designs and API selections.

Addressing these questions, we first consider various text re-ranking prompts tailored to accomplish the re-ranking task within a budget. Next, we explore different LLM APIs and their associated costs for implementing these prompts. We then introduce a novel two-layer approach, EcoRank (depicted in Figure 4.1), which begins by re-ranking initially ranked passages (e.g., those ranked using BM25) with *pointwise relevance filtering* on a *high-accuracy* (and consequently *expensive*) LLM API, utilizing a fraction of our budget. This initial re-ranking demotes irrelevant passages, allowing us to allocate the remaining budget to re-rank relatively relevant passages. In the second layer, we use a *less accurate* (and thus *cheaper*) LLM API, applying the remaining budget to further re-rank the passages using *pairwise ranking prompting*. We evaluate various single or hybrid (combining more than one) prompt designs for the text re-ranking problem, for various budgets and APIs on four popular datasets: Natural Questions (NQ) [86], Web Questions (WQ) [24], TREC [48] DL19, and DL20. Our most efficient method EcoRank achieves a gain of 14% on MRR and R@1 ranking accuracy than baselines. Our contributions are summarized as follows.

- We introduce the problem of budget-constrained text re-ranking that considers the cost of various LLM APIs.
- We propose and compare various ranking prompt designs and API choices in a budget-constrained scenario for text re-ranking.
- We further propose a novel two-layer cascading pipeline EcoRank which optimizes the budget usage.
- We extensively evaluate and compare various prompt designs and API choices on four datasets. We make our code available to the research community. [1]

4.2 Problem Definition

We focus on optimizing passage re-ranking under budget constraints, ranking top-k passages from a pre-ranked list. These lists are often obtained from a retriever in response to a natural language query. Given a budget β, a query q, and a list of pre-ranked N passages $p_0 \cdots p_N$ with respect to q, the task is to re-rank the passages using available LLM APIs and retrieve top-k passages. For instance, in question answering tasks, BM25 is commonly used to produce the initial ranking, and then the main focus is typically on the top re-ranked passage (i.e. $k = 1$). The following designs are critical for budget-aware text re-ranking.

API choice. Assume there are M different LLM APIs available, denoted as $\mathcal{L}_1 \cdots \mathcal{L}_M$. Each API $\mathcal{L}$ takes a prompt ρ and generates an output Θ. Associated with calling each

[1] `https://github.com/shihabrashid-ucr/EcoRank`

API is a cost $\mathcal{C}$, defined in Equation 4.1:

$$\mathcal{C} = c_p \cdot len(\rho) + c_o \cdot len(\Theta) + c_f \tag{4.1}$$

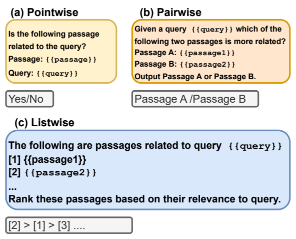
where c_p represents the cost per input or prompt token, c_o is the cost determined by the number of tokens generated by $\mathcal{L}$, and occasionally, a fixed API call cost c_f applies.

Choice of ranking prompt. The other key design choice is the set of prompts $\rho \in T$ for a text re-ranking task. Specifically, given the passages $p_0 \cdots p_N$ and query q, we need to generate a sequence of prompts, where each prompt includes the query and one or more of the input passages. For example, a prompt can ask if a passage is relevant or ask to compare two passages. As input tokens determines the cost to generate an output, choosing the right prompt(s) is very crucial.

Split of budget. Our experimental setup also accounts for scenarios where multiple prompts or rounds of iterations are required. In such cases, there is an additional factor in budget considerations: the budget β can be divided and allocated across multiple prompts ρ, $\beta \triangleq \beta \cdot x + \cdots + \beta \cdot y$ where the coefficients $x + \cdots + y = 1$.

Budget-aware optimization. Given the problem setups described above, the task of budget-aware text re-ranking essentially becomes an optimization task. The objective is to maximize the re-ranking performance, denoted as $\mathbb{E}$, subject to a given budget β. This optimization occurs within the search spaces of prompts, APIs, and budget allocations, $\mathbb{E}_{q,p}[c(\mathcal{L}, \rho)] \leq \beta$, where $c(\mathcal{L}, \rho)$ is the associated cost for processing query q with prompt ρ and LLM $\mathcal{L}$. Given the vast range of available APIs, prompts, and permutations of budget splits, this optimization presents non-trivial and unique challenges. No other research work has considered the cost of the LLMs for text re-ranking.

4.3 Budget-Aware Ranking Prompt Designs



```
Is the following passage
related to the query?
Passage: {{passage}}
Query: {{query}}
```

```
Yes/No
```

```
Given a query {{query}} which of the
following two passages is more related?
Passage A: {{passage1}}
Passage B: {{passage2}}
Output Passage A or Passage B.
```

```
Passage A /Passage B
```

```
The following are passages related to query {{query}}
[1] {{passage1}}
[2] {{passage2}}
...
Rank these passages based on their relevance to query.
```

```
[2] > [1] > [3] ....
```

Figure 4.2: Different prompt strategies for text re-ranking.

We build on previous works on ranking prompt designs. Our key contribution is making these designs budget-constrained. The different designs are depicted in Figure 4.2.

4.3.1 Pointwise Methods

Pointwise approaches process the passages one by one along with the query as a prompt. We define three types of prompts within pointwise methods.

Query generation. [142] proposed an unsupervised pipeline *UPR* to re-rank passages by asking the LLM to generate a query q' given each passage p_i from the initial ranked list of passages. Their approach is not applicable to generation-only LLMs like GPT-3 or GPT-4 which do not score the outputs. For budget constrained scenarios, we adapt their approach to generation only LLMs by asking the LLM to generate a query given a passage and measure the token-level F1 score between the newly generated query q' and original query

q and sort the passages based on this score. We start from the top of the list and go down until we exhaust the budget, and keep the rest in their original ranking positions. We call this approach `B-UPR`.

Binary Classification. We propose another type of budget-aware pointwise prompt design inspired from [96] where we ask the LLM to predict *Yes* or *No* given each passage from the initial list whether it is relevant to the query q. We ask the LLM to output only "Yes" or "No" which restricts the number of output tokens to be 1. This method can be categorized as a *coarse-grained* strategy where we group the passages based on relevance but each individual passage ranking relies on the initial score.

Likert Classification. Instead of classifying each passage in a binary fashion, we also introduce a design to categorically classify each passage into a 3-point Likert scale, inspired from [182], where we ask the LLM to classify a passage into either of the following three groups: *Very related, Somewhat related* or *Unrelated.*

4.3.2 Listwise Methods

In this strategy, passages $p_0 \cdots p_N$ are put through the LLMs with identifiers such as ([1], [2], etc.) as a list along with the query q. The LLM is then asked to give the relative ordering of the passages as an output (i.e. [2] ¿ [1] $\cdots$). Recent works like RankGPT [154], LRL [102] introduce a sliding window strategy to combat token limitation challenge in LLMs, where a sliding window of size w with a step of s is used. For a budget-constrained scenario, we approximate the number of passages that can be given as input to the prompt. We call this `B-RankGPT`.

LLM	Parameters	Cost[‡]	Provided By	Methods (MRR)			Methods (R@1)		
				Pointwise	PRP	Listwise	Pointwise	PRP	Listwise
FLAN T5-XL	3B	1x	Replicate	42.08	**46.54**	NA	33.4	**39.5**	NA
FLAN T5-L	800M	$\frac{1}{3}$x	Replicate	37.91	38.03	NA	27.4	29.2	NA
Llama2	7B	1x	TextSynth	30.24	21.18	NA	20.2	10.6	NA
Falcon	7B	1x	TextSynth	30.8	23.44	NA	20.6	12.8	NA
GPT-curie	6.7B	5x	OpenAI	30.61	19.52	NA	20.8	9.40	NA
GPT-3.5-turbo	175B	10x	OpenAI	34.41	43.82	42.3	23.9	36.60	34.3

Table 4.1: Different LLMs' performance with various strategies on a subset of NQ dataset for top-20 passages. [‡] Cost is measured as a unit here as the pricing may vary with time.

However, as the prompt of listwise method is rather complicated (LLMs have to understand the ordering of multiple passages and have to give an output in a structured format), most LLM APIs face issues in giving the correct output. [124] show that medium-sized LLM APIs like FLAN-T5-XL [42] are inconsistent and not able to understand the prompt correctly and provide irrelevant results. Only big-sized commercial LLMs like GPT-4 and GPT-3.5 are able to utilize this approach correctly. Further, it is highly sensitive to input ordering, meaning the output depends heavily on the order of the passages in the prompt.

4.3.3 Pairwise Methods

This is a *fine-grained* strategy where each passage is compared with each other similar to bubble-sort and their ranking is modified. Given two passages p_i, p_j, and one query q, the LLM is asked to choose one passage which is more relevant to the query. This ensures the *relative* ordering among the passages. [124] proposed various pairwise ranking prompts (PRP). Among them, *PRP-Sliding*, which does k rounds of bubble-sort pass that ensures the top-k ranking performs best. However, all the approaches, even sliding-k are quite expensive. To get top-k ranking, $N \cdot k$ API calls need to be made with each call having approximately twice the number of tokens (because two passages per prompt) compared to pointwise methods, which need to make N calls with each call having less tokens. In contrast, PRP methods can achieve fine-grained ranking accuracy. We adapt this strategy to a budget-constrained one by approximating the number of calls that could be made within the budget. If τ such calls can be made, we start at $l = min(k, \tau)$-th positioned passage in the initial ranked list and move the passage up the list. We continue iterating until $\tau = 0$. For passages that could not be processed, we take their initial ranking position. We call this approach `B-PRP`.

4.4 EcoRank

In this section, we present our novel most efficient budget-aware approach shown in Figure 4.1. Both listwise and pairwise methods can achieve accurate rankings but they suffer from high cost. Listwise methods only work with very expensive LLMs hence we put our focus on pairwise approach. While constrained within a budget, the number of passages that can be processed by LLMs impacts the final ranking accuracy. Cheaper LLMs are able to process more passages but they lack quality. Therefore, there are two key challenges that needs to be solved: 1) how to ensure quality 2) how to ensure quantity. With no budget constraint, pairwise methods may be an obvious choice due to their fine-grained accuracy but they have several limitations. We solve the challenges of pairwise designs in a budget-constrained scenario in a two-staged fashion as shown in Figure 4.1.

First stage. Only the first few passages can be compared by pairwise with a limited budget. However, as the initial ranked list is not that good, valuable tokens may be spent on irrelevant passages. Thus in the first stage, we intelligently pick the passages to do pairwise comparison on.

We split our budget β into two fractions x and y and use x amount to filter the passages using binary classification approach from the pointwise prompting group. We use a stronger and comparatively expensive LLM $\mathcal{L}_1$ with cost $\mathcal{C}_1$ to generate a relevance for the passages in the form of "Yes" or "No" and create an intermediate ranked list as per the binary classification strategy mentioned above. This coarse-grained strategy ensures *quality* as we are using a strong LLM to put more relevant passages at the top for the next stage and push irrelevant ones to the bottom of the list.

Second stage. We spend the rest y amount using pairwise prompting design to compare two passages at a time. To ensure *quantity*, we use a cheaper LLM API $\mathcal{L}_2$ with cost $\mathcal{C}_2$ to do the comparisons. As the passages have already been filtered by a stronger LLM, a cheaper LLM does not hinder the quality that much rather it can process $(\mathcal{C}_1/\mathcal{C}_2)$x the number of passages than the expensive variant for a fixed budget. Further, as there are much fewer "Yes" passages compared to "No", the unprocessed passages from first stage can be processed at this stage.

While further stages can be added to the pipeline, we see diminishing results as we add more stages. Thus we choose two.

Choosing the LLMs. As seen in Table 4.1, LLMs' performance is not proportional to their cost. Some expensive API (i.e. GPT-curie or GPT-3.5) may be less effective in zero-shot text re-ranking tasks than comparatively cheaper LLM (i.e. T5-L or T5-XL). Further, same-priced LLMs may not give the same performance. A key optimization here is to choose the appropriate LLM which will perform the task accurately without much cost. As the T5 LLMs perform significantly better with reasonable cost, we choose Flan T5-XL as the costlier API $\mathcal{L}_1$ and Flan T5-L as the cheaper API $\mathcal{L}_2$ in EcoRank.

Optimization of budget split. Another key challenge here is the split of budget x and y. Putting more budget in the first stage will ensure more *quality* filtering whereas putting more budget on the second stage will do more pairwise but on less relevant passages. We hypothesize that both stages contribute equally to the pipeline and choose an equal split of x and y.

4.5 Experimental Evaluation

4.5.1 Setup

Implementation Details. We host the LLMs offline (except for GPT3.5) and define the budget in terms of number of tokens. The GPU instance we use is g5.4xlarge. The prices of LLMs change with time frequently so we make a standard approximation of the costs of each LLM shown in Table 4.1. For example, T5-L is approximated as 3x cheaper than T5-XL and GPT-3.5 is 10x costlier than T5-XL based on the pricing page of different services [4, 6, 7].

To implement the supervised models, we load the available pre-trained versions in our GPU and use them to generate embeddings of the passages and queries. We use PyGaggle [2] to re-rank the passages with the loaded model.

To implement InPars, We use GPT-3.5 to generate synthetic data d fully using our budget B1, B2, B3. Using B1 we could generate around 50K, using B2 10K, and using B3 5K questions. Thus we have three sets of training data d_{b1}, d_{b2}, and d_{b3}. We randomly sampled passages from the corpus and asked GPT-3.5 to generate a new question. These are our positive examples. To get the negative examples, we use BM25 to retrieve 5 relevant passages. The passages that are not gold passages are considered as negative examples. We trained the T5-large [3] models for three budget categories with 156 steps and the same training arguments as InPars. We use the corresponding trained model to generate embeddings of the passages and re-rank.

Datasets. Following previous work on passage retrieval, we choose the popular benchmark datasets Natural Questions (NQ) [86], Web Questions (WQ) [24], TREC [48] DL19, and DL20. There are total of 3610 questions on NQ, 2032 on WQ, 30 on DL19, and 44 on DL20 test splits. For TREC datasets, there are multiple relevant passages per query contrary to NQ and WQ. To ensure fairness among datasets, we consider the passages with a score of 3 to be the relevant ones for TREC.

Budget categories. We experiment with three budget categories. Budgets are represented as token limits per question. The number of tokens that can be processed with each

[2]https://github.com/castorini/pygaggle
[3]https://huggingface.co/google-t5/t5-large

Category	Cost	T5-XL	T5-L	GPT-3.5
B1	0.57c	20000	60000	2000
B2	0.11c	4000	12000	400
B3	0.05c	2000	6000	200

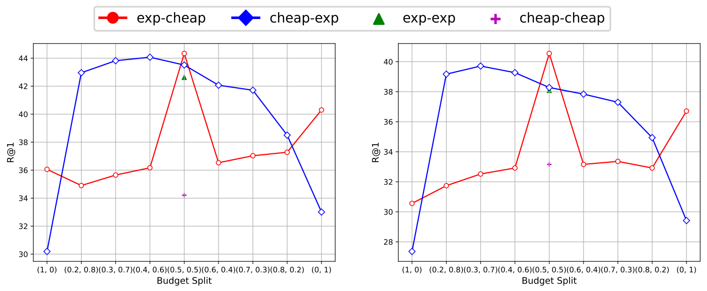
Table 4.2: Budget categories for different LLMs.

LLM is mentioned in Table 4.2. We choose a relatively higher budget B1 which can process most passages with pointwise and complete one pass with pairwise, and lower budgets B2, B3 where only some passages can be processed to show the efficacy of our solutions.

Baselines. On top of the budget-constrained methods introduced above, we show comparisons with state-of-the-art supervised and unsupervised baselines. The supervised baselines are: 1) **monoBERT** [117]: A cross-encoder reranker trained on BERT-large, trained on MSMARCO, 2) **monoT5** [118]: A sequence-to-sequence reranker based on T5, and 3) **TART** [20]: A supervised instruction-tuned passage reranker based on FLAN-T5-XL

For unsupervised approach, we consider **InPars** [27], where we generate synthetic training data using GPT-3 and use them to train a T5-large model. For each budget category, we spend all our budget to generate synthetic data and infer the trained model in a zero-shot approach. We also consider OpenAI *text-ada-002* embedding model as a reranker baseline.

We assume zero cost for the supervised baselines and no inference cost for InPars as no paid API is used. In reality, supervised models can have some costs regarding time and computing resources.



(a) Dataset NQ (b) Dataset WQ

Figure 4.3: (a) and (b) subgraphs show the impact of our parameter choices in EcoRank for budget B2.

4.5.2 Main Results

We show our main evaluation results for different budgets and ranking strategies in Table 4.3 for $N = 50$ passages. We choose the popular Mean Reciprocal Rank (MRR) and Recall@k as our evaluation metrics following previous work. All LLM-based approaches increase the initial ranking significantly except for B-UPR. We see that, overall, for all budget categories, EcoRank surpasses all other approaches, even the supervised ones. For TREC DL 20 dataset, as there are many relevant passages given a query, B-PRP performs a little better for budget B1 and ours performs similarly. B-RankGPT is promising but it can only work with a high budget (i.e. B1) as GPT-3.5 is expensive.

Supervised vs Unsupervised. We see that supervised models sometimes perform better than EcoRank in some datasets in B3. As we assume zero cost for supervised models, *budget* is not directly applicable. They can process all the passages while budget-aware approaches can process only a few in B3. Even with this limitation, our methods perform better than supervised models in B1 and B2. Further, supervised models have the following restrictions: 1) training data may either not be available or be very difficult to collect, especially if the domain is niche, and 2) training data may incur high annotation costs. In real life systems, they are expected to perform worse. Unsupervised fine-tuning based approaches like InPars also do not perform as well as ours.

High to low budget analysis. Among budget-aware methods, for higher budget (B1, B2), we see B-PRP method performing better than pointwise methods like binary classification in MRR and R@1 metrics. As we decrease our budget (B3), due to pairwise methods not processing enough passages, they perform worse than binary method in MRR. EcoRank achieves the best results in all budget categories, proving the efficiency of this approach. The gain of EcoRank with the second best approach increases from an average of 2% for higher budget to 12% for lower budget for R@1. This shows with lower budget constraint, our most efficient approach can perform really well.

4.5.3 Analysis of our chosen parameters

For EcoRank we have made some decisions regarding three sets of parameters: 1) The choices of prompting strategies, 2) The choices of expensive and cheap LLMs $\mathcal{L}_1$, $\mathcal{L}_2$, and 3) The choices of budget split between the prompts x and y. We chose pointwise and pairwise strategies due to their *cost-to-performance ratio* as seen in Table 4.1. We choose an expensive LLM for the first stage and a cheap LLM for the second stage with equal budget split $x = 0.5$, $y = 0.5$. The justification being that an expensive LLM is needed in the first stage to filter the important passages for the pairwise approach to focus on the second stage. As both stages play an equal part, an equal budget split is the appropriate choice.

We accompany our choices with an extensive evaluation performed on two datasets with all combinations of budget splits and LLM choices shown in Figure 4.3. The blue line shows the results if we use the cheaper LLM in the first stage. We see a declining performance than choosing an expensive LLM (shown in red) in stage one. As we move closer to an equal split, the performance keeps increasing till it reaches a peak and declines again as we move away. For detailed results on chosen parameters, please refer to appendix **??**.

4.5.4 Ablation studies of EcoRank

In EcoRank, there are two main choices that impact the performance.

1) Hybrid prompt design. We use a combination of pointwise and pairwise methods. If we consider the *intermediate ranked list* (from Figure 4.1) that is generated after the first stage as the final ranking, the performance decreases as only half budget is used. Further, if we only use one ranking prompt design (either Binary or B-PRP in Table 4.3) using full budget we also see a significant decrease in performance than EcoRank. Hybrid prompt design is crucial to get the maximum performance in a budget-constrained scenario.

2) Cascading of LLMs. We show the results of a variant of EcoRank where we do not cascade LLMs but keep the hybrid prompt design. We call this approach EcoRank-w/o-cascade. Only the expensive LLM is queried in both stages. Although an expensive LLM is more accurate, we see from Table 4.3 that, this variant falls short of EcoRank. It still performs better than other methods. Using a cheaper LLM in the second stage can enable processing more passages resulting in better performance. Cascading of cheap and expensive LLMs is impactful to getting the maximum performance.

4.6 Related Work

To the best of our knowledge, we are the first to work on budget-constrained text re-ranking problem with LLMs. We divide the related work into LLMs in text re-ranking and Cost aware LLMs.

LLMs in text re-ranking. There are three main zero-shot prompting strategies to re-rank initially ranked passages. They are pointwise [96, 142, 182], listwise [102, 154, 155], and pairwise [124], which we covered in details in Section 4.3. Each strategy has their own strengths and may not work with all types of LLMs. Very recently another prompting strategy has been introduced called *setwise* [183], which is an improvement over listwise approach where instead of outputting an ordered list of documents, a single document which is the most relevant is given as output. Although this reduces the computational overhead of listwise and pairwise methods, it works best with LLMs which can output scores of generation. Other works like distillation [153], RankVicuna [122], RankZephyr [123] train an open source model with training data to improve listwise approaches. All these approaches do not consider the cost of LLMs and do not try to optimize the performance of text rankers with a budget constrain. Prior to recent efforts with LLMs in text re-ranking, most works focused on the supervised ranking problem using monoT5 [118] or BERT [181] where they trained a pre-trained LM (PLM) for re-ranking tasks. Other supervised methods focus on generating data to train PLMs like InPars [27], Promptagator [50], ExaRanker [61], SPTAR [121], HyDE [63] etc. They mainly use LLMs as an auxiliary tool to support the training of PLMs and thus different from the scope of this chapter.

Cost-aware LLMs. There are some works which focus on cost-aware applications of LLMs but in other areas than text re-ranking. FrugalGPT [37] uses LLM cascading to reduce the cost of API calls but they require a trained model to score the generation quality similar to FORC [145] which uses a trained meta-model to predict performance of LLMs. These trained models require fine-tuning data which may be difficult to obtain. MoT [173] uses answer sampling strategy which is not applicable in text re-ranking as LLMs output fixed tokens instead of open-ended. Other works focus on optimizing API calls by using a neural caching system with a student model [133]. None of these apply to our problem statement as we aim to optimize the performance in text re-ranking in a fully unsupervised fashion.

4.7 Conclusion

We contribute to the first efforts of budget-constrained text re-ranking with LLMs and have identified that existing works fail to consider budget while optimizing performance in text re-ranking. We propose a suite of budget-constrained methods with various ranking prompt designs and LLMs and extensively evaluate them on four datasets. Our most efficient method EcoRank, which is a two-layered pipeline that optimizes vast spaces of decisions, achieves a gain of 14% on MRR and R@1 than other approaches.

Method	Strategy	LLM	NQ		WQ		TREC DL 19		TREC DL 20	
			MRR	R@1	MRR	R@1	MRR	R@1	MRR	R@1
BM25	-	-	32.49	22.10	29.69	18.89	48.15	30.00	69.67	56.86
Supervised Models										
monot5-base	-	T5-base	48.01	39.11	43.15	33.21	**74.57**	**66.66**	79.94	68.18
monot5-3B	-	T5-3B	**50.47**	**41.66**	**44.88**	**35.23**	72.75	63.33	79.49	65.9
monoBERT	-	BERT	47.31	38.25	43.63	33.8	73.68	66.66	**81.06**	**70.45**
TART	-	T5-XL	46.42	37.47	42.65	33.07	73.13	60.00	68.03	52.28
Unsupervised: B1 - 0.57 cents per question										
InPars	-	T5-large	42.73	32.43	41.49	31.05	59.63	46.66	74.72	61.36
Binary	Point	T5-XL	46.77	37.36	42.09	31.54	64.86	50.00	70.67	54.54
Binary	Point	GPT-3.5	36.10	25.80	34.06	22.58	53.76	40.00	59.99	45.45
Likert	Point	T5-XL	39.49	28.25	39.81	28.44	59.86	43.33	63.85	50.00
B-UPR	Point	T5-XL	27.37	17.10	29.69	18.89	26.17	10.00	36.12	25.00
B-PRP	Pair	T5-XL	51.85	45.04	48.11	41.14	65.40	56.66	78.97	68.18
B-PRP	Pair	GPT-3.5	36.68	29.00	37.25	30.01	63.85	53.33	66.4	54.54
B-RankGPT	List	GPT-3.5	45.05	37.83	42.01	34.10	66.82	56.66	72.23	63.63
EcoRank†	Hybr.	T5-XL	**52.76**	**45.87**	**48.94**	**41.58**	**78.87**	**67.44**	**80.63**	**70.45**
Unsupervised: B2 - 0.11 cents per question										
OpenAI Embedding	-	text-ada-002	32.37	22.10	29.69	18.89	48.15	30.00	58.67	45.45
InPars	-	T5-large	43.92	33.57	42.11	31.39	60.39	46.66	72.59	54.54
Binary	Point	T5-XL	44.88	36.06	40.84	30.56	64.66	50.00	70.67	54.54
Likert	Point	T5-XL	38.96	28.00	39.17	27.95	59.86	43.33	63.78	50.00
B-UPR	Point	T5-XL	28.03	17.45	29.69	18.89	25.62	10.00	35.96	25.00
B-PRP	Pair	T5-XL	45.97	40.30	42.79	36.71	65.33	**56.66**	74.44	59.09
EcoRank-w/o-casc.	Hybr.	T5-XL	48.56	42.63	44.87	38.09	63.13	50.00	74.06	59.09
EcoRank	Hybr.	T5-XL+L	**50.72**	**44.34**	**47.05**	**40.55**	**65.58**	53.33	**80.22**	**70.45**
Unsupervised: B3 - 0.05 cents per question										
InPars	-	T5-large	44.49	34.79	42.63	32.72	59.84	46.66	74.3	61.36
Binary	Point	T5-XL	43.06	34.59	39.20	29.42	62.97	50.00	70.67	54.54
Likert	Point	T5-XL	38.31	27.83	37.65	26.91	59.79	43.33	63.58	50.00
B-UPR	Point	T5-XL	29.53	18.55	30.10	19.43	33.2	16.66	40.52	25.00
B-PRP	Pair	T5-XL	42.81	36.98	39.45	32.66	64.02	53.33	**78.93**	68.18
EcoRank-w/o-casc.	Hybr.	T5-XL	44.13	37.89	40.66	33.85	63.13	50.00	77.1	67.81
EcoRank	Hybr.	T5-XL+L	**46.83**	**40.33**	**43.86**	**37.00**	**66.92**	**56.66**	78.76	**70.45**

Table 4.3: Results (MRR and R@1) on all datasets for 50 passages. For B2 and B3, the budget is too low for B-RankGPT to have any impact hence it is omitted.

Chapter 5

Iterative Query Expansion for Retrieval Over Cost-constrained Data Sources

5.1 Introduction

Many Information Retrieval and AI tasks depend on the availability of an effective *Retriever* module. A *Retriever* extracts k relevant documents or passages, given a query. It serves as a standalone task as a core component in modern search engines, or as an intermediate step for retrieval-augmented question-answering or other downstream tasks. There are two main paradigms for retrievers: 1) *sparse* or lexical-based retrievers such as BM25 [139], and 2) *dense* or embedding-based retrievers like DPR [79] and Contriever [72]. Dense retrievers have been shown to perform better when large amounts of labeled data are available, whereas BM25 remains competitive on out-of-domain datasets [156].

Query expansion is a popular approach to improve the accuracy of retrievers []. A popular method to expand the query has been the use of pseudo-relevance-feedback (PRF) [141, 175], which addresses the query-to-document vocabulary mismatch problem. Key terms are extracted from the top-k relevant documents in the first pass retrieval and appended to the original query to perform the final retrieval. However, the documents returned from the first stage retrieval may not be relevant, and may introduce noise thus hindering the effectiveness of PRF. To alleviate this issue, recent LLM-based approaches like query2doc [162], CoT [73] and GRF [104] skip the first-stage retrieval and use LLMs to generate additional content to append to the original query. These approaches use pre-trained LLMs as black boxes and have shown improved results.

A salient assumption of these LLM query expansion works is that the cost of retrieving documents is low, compared to the cost of accessing the LLM. For example, the document collection may be stored in Elastic Search, which has a very low per-query cost. We argue that this assumption is not true in several important problem settings, where the dominant cost is the retrieval of result documents. This is the case when the document corpus is not available or indexed locally, but is accessed via APIs. For example, legal document retrieval systems like PACER [9], Westlaw [10] and LexisNexis [8] charge a fee for retrieving each document. These fees can be as high as 0.1 USD per page of a document [9].

Our key idea for improving the retrieval accuracy is that we combine classic pseudo-relevance feedback expansion techniques with modern LLM-based query expansion tech-

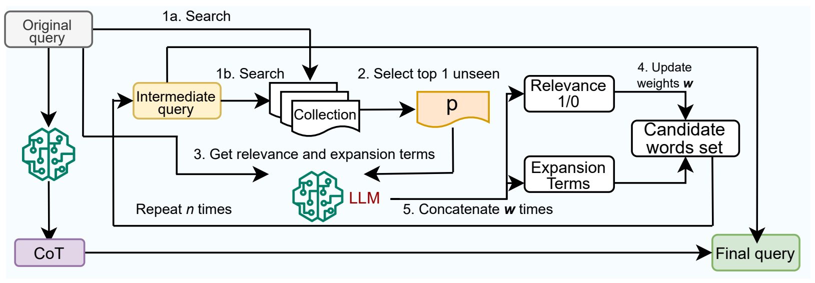


Figure 5.1: An overview of our proposed solution

niques. To mitigate the drawbacks of pseudo relevance feedback, we employ an LLM as a relevance judge for each returned result. Specifically, we propose ProQE, shown in Figure 5.1, which is a *progressive* query expansion algorithm that iteratively expands the query as it retrieves more documents.

Creating a progressive query expansion algorithms is challenging for several reasons. First, we need to decide how the terms in each retrieved document, whose relevance is uncertain, should be used to potentially adapt the query. This is related to the exploration vs. exploitation paradigm: if we retrieve more documents using the original query, we may view more diverse results, whereas aggressively refining the query using the early retrieved documents may improve the focus and accuracy of the retrieval. Second, we want to create an expansion method that performs well for most of the popular ranking algorithms, both based on sparse and dense retrieval. This will allow our method to be applicable to a wide range of black-box (e.g. API-based) ranking systems which may input a list of keywords or weights and return a ranked list of results.

A key feature of ProQE is its plug-and-play capability, allowing it to integrate seamlessly with any sparse or dense retrieval methods. The process operates as follows.

For Sparse retrievals, we first retrieve the top one new document using the original query. Through two LLM calls, it extracts potential expansion terms from this document and further scores the relevance. Our scoring function takes the relevance score and all previously retrieved terms as input, and updates the weights. The terms are appended to the query based on their updated weights. We repeat this process n times. Retrieving only 1 new document at each iteration helps by saving unnecessary retrieval costs. Further, evaluating each document ensures that terms from more relevant documents receive higher weights, allowing for progressive updates to the query terms based on LLM feedback. Finally, after n iterations, the final query is formulated by prompting the LLM using chain-of-thought [166] to retrieve additional context and appending it to the intermediate query. For dense retrieval models, separate embeddings of the original query, expansion terms, and CoT output are created and then combined using a weighted average to form the final query embedding.

We extensively compare our method to state-of-the-art pseudo relevance feedback and generative query expansion approaches, for multiple types of sparse and dense retrieval models on four popular datasets: Natural Questions (NQ) [86], Web Questions (WQ) [24], TREC [48] DL19, and DL20. ProQE achieves an average gain of 37% on MRR and R@1 ranking accuracy compared to the baselines.

Our contributions are summarized as follows:

- We introduce the problem of LLM-assisted retrieval over cost-constrained black-box data sources.
- We propose novel progressive query expansion algorithms for both sparse and dense retrieval systems.

- We extensively evaluate and compare various baseline expansion methods, over various retrieval models on four datasets. We make our code available to the research community.[1]

5.2 ProQE: Progressive Query Expansion

Problem Definition. We focus on retrieving k documents from a collection $\mathcal{D}$, either with Sparse or Dense retrieval approaches. The query expansion task is formulated as generating an expanded query q' that contains additional query terms that may help in retrieving relevant documents, given the original query q. We assume $\mathcal{D}$ is not indexed locally and is only accessible via a retrieval API $\mathcal{A}$. $\mathcal{A}$ charges a cost $\mathcal{C}$ for the retrieval of each new document given a query. The cost typically does not depend on any variable, i.e. the size of the query. Note that we assume that the query interface only charges for retrieving new unique documents. That is, if we expand the query and resubmit and the same document p is returned, there is no additional cost. This is not a necessary assumption for ProQE, but it is commonly used by commercial systems such as ScrapeOps web content retriever [11].

ProQE extracts key terms from each retrieved document and uses these terms at each iteration to modify the query, before retrieving more documents. We next discuss the details of ProQE for sparse and dense retrieval systems.

[1]link here

ProQE for Sparse Retrieval. Our method first retrieves top-1 new document p_1 calling $\mathcal{A}$ using the original query q. The relevance of the passage $rel(p_i)$ is assessed by prompting an LLM $\mathcal{L}$ with a pointwise ranking instruction [96], *"Is the following passage related to the query?"*. In parallel, m potential expansion terms are extracted using $\mathcal{L}$ with q and p_1 given as input with the instruction *"Given the query and passage, extract 5 keywords that may be useful to better retrieve relevant passages."*. The weights $w(t_i)$ of terms $t_1 \cdots t_m$ are updated using the following equation and kept in a global dictionary with total terms M.

$$w(t_i) = \begin{cases} w(t_i) + \beta, & \text{if } rel(p_1) = 1 \\ w(t_i) - \gamma, & \text{if } rel(p_1) = 0 \end{cases} \tag{5.1}$$

The terms with $w(t_i) > 0$ are considered as expansion terms and are repeated $int(w(t_i))$ times and appended to the original query. The original query is boosted α times to form the intermediate query q^+.

$$q^+ = concat(\{q\} \times \alpha, \sum_i^M \{t_i\} \times w(t_i)) \tag{5.2}$$

This process is iterated n times. At the beginning of each iteration, the intermediate query q^+ is used to retrieve p_1 and a new q^+ is generated at the end. By boosting and decreasing the weights of expansion terms based on feedback from the LLM and the retrieved passage, only relevant terms are appended to the query, thereby reducing noise. The iterative process facilitates focused retrieval in each turn, leading to the generation of effective query terms. Any irrelevant term added in one iteration is corrected in subsequent iterations. We tune m, n, α, β, and γ on dev sets and show that the number of iterations does not vary the final performance much. We discuss parameter details further in Section 3.4.

Finally, after n-th iteration, we prompt $\mathcal{L}$ using chain-of-thought instruction [73]: *"Answer the following query, give rationale before answering."* and receive the output θ_c. We stop the iteration at n as further updates do not improve the performance and may deteriorate. The final query q' is formulated as $q' = concat(q^+, \theta_c)$. We observed that appending θ_c with q at the start of the iterations adversely impacts performance, as the non-factual outputs from the LLM can misdirect the progressive update of queries via relevant passages.

Dense Retrieval. Query expansion with key terms typically works best for sparse retrievals as expansion targets vocabulary mismatch and is uncommon for API-based retrieval systems. Nonetheless, for completeness, we show that ProQE also improves the dense retrieval system. Appending a term multiple times does not boost its weight in a dense retrieval system as the whole semantic meaning is captured in an embedding. We use an encoder from a dense retriever model to create embeddings for the original query $\vec{\mathcal{E}_q}$. After each iteration, intermediate query embedding $\vec{\mathcal{E}_{q^+}}$ is computed as follows.

$$\vec{\mathcal{E}_{q^+}} = \sigma \times \vec{\mathcal{E}_q} + \tau \times \frac{1}{M} \sum_{i}^{M} w(t_i) \times \vec{\mathcal{E}_{ti}} \tag{5.3}$$

where σ is the query weight and τ is the term weight for dense models. After n iterations, similarly, we create the embedding for the CoT output $\vec{\mathcal{E}_{\theta c}}$ and compute the final query embedding $\vec{\mathcal{E}_{q'}} = \sigma \times \vec{\mathcal{E}_{q^+}} + \delta \times \vec{\mathcal{E}_{\theta c}}$, where δ is the CoT weight. We use this final query embedding to search the corpus embeddings using similarity search to retrieve the documents.

5.3 Experimental Evaluation

Datasets. Following previous work on passage retrieval, we choose the popular benchmark datasets Natural Questions (NQ) [86], Web Questions (WQ) [24], TREC [48] DL19, and DL20. For TREC datasets, there are multiple relevant passages per query contrary to NQ and WQ. To ensure fairness among datasets, we consider the passages with a score of 3 to be the relevant ones.

Implementation. For experiments, we indexed the document corpus with Pyserini. For LLM choice, we compared with GPT-3.5, Flan T5-XL, Llama-2 on dev set and chose T5-XL as it has the best cost-to-performance ratio. This choice is also supported by previous work [73,135]. We tuned our sparse weight parameters α, β, γ with a range from 0 to 5 and step size of 1, dense weight parameters σ, τ, and δ with a range from 0 to 1 and step size of 0.1, iteration number n (range from 2 to 15 with step size of 1), and number of potential expansion terms m (range from 3 to 7 with step size of 1) on the dev sets of our datasets and chose the values $\alpha = 1$, $\beta = 1$, $\gamma = 0$, $\sigma = 0.8$, $\tau = 0.2$, $\delta = 0.2$, $n = 5$, and $m = 5$. Note that, our choices of α, σ, τ, δ, and m are also supported by previous work [104,162].

Baselines. We sampled from each retrieval category, sparse and dense with unsupervised and supervised variants to show the effectiveness of ProQE. For sparse retrieval, we compare with BM25 and docT5 [119] as retrievers. docT5 uses a trained *T5-large* model to generate a query given a document and the generated query is appended at the end of the document.

Method	NQ		WQ		TREC DL 19		TREC DL 20	
	MRR	R@1	MRR	R@1	MRR	R@1	MRR	R@1
Sparse Retrieval								
BM25	29.84	20.77	28.16	19.00	33.59	20.93	12.85	10.00
+RM3	28.76	20.24	31.16	22.78	30.09	20.93	10.89	8.00
+Rocchio PRF	25.42	17.61	26.28	18.65	28.33	18.60	10.40	8.00
+query2doc ZS	32.65	24.73	38.05	30.41	26.44	13.95	11.02	8.50
+query2doc FS	35.11	25.70	38.58	29.13	**37.03**	20.93	12.86	10.50
+CoT	35.42	26.48	44.07	35.48	35.78	25.58	13.73	11.00
+GRF	33.51	26.34	42.22	34.99	23.31	11.62	10.69	8.50
+NORMY	**39.48**	**33.01**	**47.70**	**40.89**	34.12	**27.90**	**14.71**	**12.50**
docT5	-	-	-	-	**44.87**	32.55	13.96	10.50
+NORMY	-	-	-	-	43.05	**34.88**	**14.02**	**13.60**
Dense Retrieval								
DPR	22.67	10.33	24.69	13.13	-	-	-	-
+CoT	23.13	10.55	25.29	12.99	-	-	-	-
+query2doc	23.38	11.02	25.23	13.04	-	-	-	-
+NORMY	**24.73**	**12.32**	**26.42**	**14.18**	-	-	-	-
TCT-Colbert	-	-	-	-	46.66	37.20	16.33	14.01
+ NORMY	-	-	-	-	**47.17**	**39.53**	**16.45**	**14.01**

Table 5.1: Results (MRR and R@1) on all datasets for 20 passages. Best performing are marked bold.

The document is then indexed to better match the query. We use Pyserini's prebuilt index `msmarco-v1-passage-d2q-t5`. For dense retrieval, we compare with DPR [79] and TCT-Colbert [98]. We use DPR's question encoder fine-tuned on NQ and multiset and the prebuilt index of Pyserini. TCT-Colbert fine-tunes a student encoder with distillation from a teacher ColBERT [80] model. We use `castorini /tct_colbert-msmarco` as the query encoder.

We chose both state-of-the-art pseudo-relevance feedback and generative models to compare against ProQE as comparing methods. Specifically, we choose the following:

RM3 [12]: A PRF approach that expands the query from top-k retrieved documents. We use *fb_terms = 10, fb_doc = 10*, and *query- weight = 0.5* following Pyserini's instructions [**?**].

Rocchio PRF [141]: Classic Rocchio formula with *fb_terms = 5* and *fb_docs = 3.*

query2doc [162]: Additional passage generated from queries using LLMs. We show both the zero-shot (ZS) and few-shot (FS) variants.

CoT [73]: Chain-of-Thought prompting output from LLMs.

Generative relevance feedback (GRF) [104]: Multiple types of additional content such as news articles, essays, keywords, queries, and entities generated from LLMs given the original query and combined together.

Main Results: Sparse We show our main evaluation results in Table 4.3 for $k = 20$ passages. We choose the popular Mean Reciprocal Rank (MRR) and Recall@k as our evaluation metrics following previous work. We see that in both types of retrievals, ProQE improves the baselines by up to an average of 37% from the variant without expansion. PRF methods like RM3, Rocchio do not perform well due to the top retrieved passages not being relevant. Among LLM-based generative methods, both query2doc and CoT perform really well and significantly improve the performance. However, to get the best results, feedback from both retrieved passages and LLMs are needed. The few-shot variant of query2doc has better MRR in DL 19 dataset than ours but worse R@1. Generating larger passages with LLM in some scenarios may retrieve better results at k ¿¿ 1 position. However, in Retrieval Augmented Generation tasks, only the top few passages are considered hence R@1 metric is more important. docT5 uses a fine-tuned model hence the performance is better than BM25. We see that ProQE still improves a trained model's performance.

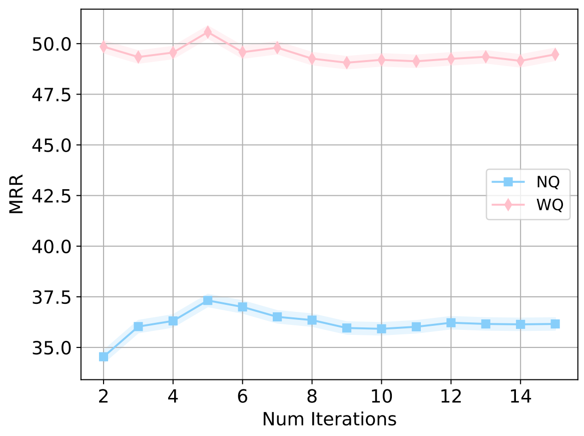


Figure 5.2: Impact of iterations for NQ and WQ dev sets.

Main Results: Dense We see that ProQE improves native dense retrievals by an average of 8%, although the margin is lower than sparse. Unsupervised dense retrieval is not typically suited for query expansion and is uncommon for cost-constrained data sources. Regardless, our method is applicable to any such system if released in the future.

Analysis: Impact of iterations. We show the impact of iteration number in Figure 5.2. We see diminishing results after 5 iterations. However, the performance does not vary much after 5. This shows that, regardless of what iteration number is chosen, it will improve the performance of native retrieval systems and other strong baselines.

5.4 Related Work

Query Expansion. To resolve the lexical mismatch between query and relevant documents, relevance feedback from documents [101, 141] or knowledge sources [51, 108, 169] are used to expand the query. In cases where the gold labels are not available, the top retrieved documents are used as pseudo-relevant documents like KL [175], RM3 [89] etc. PRF methods are primarily used in Sparse retrievals and may introduce noise in expanded terms, affecting its reliability. Recently, there have been systems for learned sparse retrievals like SPLADE [62], which is a neural retrieval model that uses BERT and sparse regularization to learn query and document sparse expansions. PRF methods have also been adapted by embedding-based dense retrieval models [79] like ANCE-PRF [172], ColBERT-PRF [165] which extracts relevant embeddings from retrieved documents to incorporate to the query embedding. Both learned sparse and dense retrieval models require training data with gold relevance labels which becomes exponentially difficult to collect if the corpus is not available locally. Further, our algorithm can work with both sparse, learned and unsupervised dense retrieval models.

LLM Augmentation. The use of LLMs [29] have spread to different augmentation techniques such as query rewriting [74,168], query-specific reasoning [61], document augmenting (doc2query) [119] etc. Some very recent LLM based query expansion works include query2doc [162], where an LLM generated document is augmented; GRF [103,104], where additional context such as keywords, news, facts generated using LLM are appended, and CoT [73], where a chain of thought answer is appended to the query. These works exclusively use LLMs as additional context which have been shown to hallucinate. Other works include HyDE [63], and GAR [19] which require trained models to compute the embeddings of the generated documents.

Cost-aware Methods. To the best of our knowledge, we are the first to consider the cost of retrieval as a constraint. Other cost-aware methods like FrugalGPT [37], Eco-

Rank [135] consider the costs of LLM APIs are used for either direct question-answering, reasoning, or text re-ranking tasks. We show that, in practical scenarios, retrieval API costs can dominate the total cost of retrieval augmented generation.

Chapter 6

Conclusions

In this dissertation, we focused on four problems related to the open retrieval question answering system. First we tackled the training data scarcity problem of frequently asked question answering. We proposed QuAX, a pipeline to automatically extract high-utility question answer pairs. Second, we focused on the open retrieval conversational question answering task. We proposed NORMY, which models the conversational history differently in each module of the pipeline. NORMY is non-uniform and performs better than all other baselines. Third, we introduced the problem of budget-constrained text reranking with Large Language Models. LLMs can be very expensive and we propose EcoRank, a budget-aware pipeline that jointly optimizes the prompt choices, LLM choices, and budget split. Finally, we focus on the Retriever component of the ORQA pipeline. We proposed ProQE, an iterative query expansion method that combines classic PRF methods with LLM-based techniques. With our novel scoring function, we showed that ProQE can adapt with any sparse or dense retrieval systems and improves the baselines.

Bibliography

[1] Ai21 - differentiate your product with generative text ai. `https://www.ai21.com/studio`. Accessed: 2023-12-10.

[2] Cohere - the leading ai platform for enterprise. `https://cohere.com/`. Accessed: 2023-12-10.

[3] doc2dial-OR dataset. `https://figshare.com/s/bf5ba94bc71b31fffdf2`.

[4] Openai api pricing. `https://openai.com/pricing`. Accessed: 2023-11-07.

[5] Predicting time and cost on mturk. https://www.cloudresearch.com/resources/blog/a-simple-formula-for-predicting-the-time-to-complete-a-study-on-mechanical-turk/. Accessed: 2024-02-14.

[6] Replicate - run open source machine learning models. `https://replicate.com/`. Accessed: 2023-11-20.

[7] Textsynth - api pricing. `https://textsynth.com/pricing.html`. Accessed: 2023-12-06.

[8] Lexisnexis: Legal and professional solutions with ai, mar 2024.

[9] Pacer: Public access to court electronic records, mar 2024.

[10] Thomson reuters: Westlaw dockets, mar 2024.

[11] Your complete toolbox for web scraping, mar 2024.

[12] Nasreen Abdul-Jaleel, James Allan, W Bruce Croft, Fernando Diaz, Leah Larkey, Xiaoyan Li, Mark D Smucker, and Courtney Wade. Umass at trec 2004: Novelty and hard. *Computer Science Department Faculty Publication Series*, page 189, 2004.

[13] Vaibhav Adlakha, Shehzaad Dhuliawala, Kaheer Suleman, Harm de Vries, and Siva Reddy. Topiocqa: Open-domain conversational question answering with topic switching. *Transactions of the Association for Computational Linguistics*, 10:468–483, 2022.

[14] Parag Agrawal, Tulasi Menon, Aya Kam, Michel Naim, Chaikesh Chouragade, Gurvinder Singh, Rohan Kulkarni, Anshuman Suri, Sahithi Katakam, Vineet Pratik, et al. Qnamaker: Data to bot in 2 minutes. In *Companion Proceedings of the Web Conference 2020*, pages 131–134, 2020.

[15] Betul Altay, Tansel Dokeroglu, and Ahmet Cosar. Context-sensitive and keyword density-based supervised machine learning techniques for malicious webpage detection. *Soft Computing*, 23(12):4177–4191, 2019.

[16] Bang An, Wenjun Wu, and Huimin Han. Deep active learning for text classification. In *Proceedings of the 2nd International Conference on Vision, Image and Signal Processing*, pages 1–6, 2018.

[17] Raviteja Anantha, Svitlana Vakulenko, Zhucheng Tu, Shayne Longpre, Stephen Pulman, and Srinivas Chappidi. Open-domain question answering goes conversational via question rewriting. *arXiv preprint arXiv:2010.04898*, 2020.

[18] Md Adnan Arefeen, Biplob Debnath, and Srimat Chakradhar. Leancontext: Cost-efficient domain-specific question answering using llms. *arXiv preprint arXiv:2309.00841*, 2023.

[19] Daman Arora, Anush Kini, Sayak Ray Chowdhury, Nagarajan Natarajan, Gaurav Sinha, and Amit Sharma. Gar-meets-rag paradigm for zero-shot information retrieval. *arXiv preprint arXiv:2310.20158*, 2023.

[20] Akari Asai, Timo Schick, Patrick Lewis, Xilun Chen, Gautier Izacard, Sebastian Riedel, Hannaneh Hajishirzi, and Wen-tau Yih. Task-aware retrieval with instructions. *arXiv preprint arXiv:2211.09260*, 2022.

[21] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

[22] Garrett Beatty, Ethan Kochis, and Michael Bloodgood. The use of unlabeled data versus labeled data for stopping active learning for text classification. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, pages 287–294. IEEE, 2019.

[23] Nicholas J Belkin, Colleen Cool, Adelheit Stein, and Ulrich Thiel. Cases, scripts, and information-seeking strategies: On the design of interactive information retrieval systems. *Expert systems with applications*, 9(3):379–395, 1995.

[24] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544, 2013.

[25] Sumit Bhatia, Debapriyo Majumdar, and Prasenjit Mitra. Query suggestions in the absence of query logs. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 795–804, 2011.

[26] Keping Bi, Qingyao Ai, Yongfeng Zhang, and W Bruce Croft. Conversational product search based on negative feedback. In *Proceedings of the 28th acm international conference on information and knowledge management*, pages 359–368, 2019.

[27] Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. Inpars: Data augmentation for information retrieval using large language models. *arXiv preprint arXiv:2202.05144*, 2022.

[28] Florian Boudin. pke: an open source python-based keyphrase extraction toolkit. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 69–73, Osaka, Japan, December 2016.

[29] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[30] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[31] Jon Ander Campos, Arantxa Otegi, Aitor Soroa, Jan Deriu, Mark Cieliebak, and Eneko Agirre. Doqa–accessing domain-specific faqs via conversational qa. *arXiv preprint arXiv:2005.01328*, 2020.

[32] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Mário Jorge, Célia Nunes, and Adam Jatowt. A text feature based automatic keyword extraction method for single documents. In *European conference on information retrieval*, pages 684–691. Springer, 2018.

[33] Claudio Carpineto and Giovanni Romano. A survey of automatic query expansion in information retrieval. *Acm Computing Surveys (CSUR)*, 44(1):1–50, 2012.

[34] Ankush Chatterjee, Manish Gupta, and Puneet Agrawal. Faqaugmenter: suggesting questions for enterprise faq pages. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 829–832, 2020.

[35] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

[36] Jiangning Chen, Heinrich Matzinger, Haoyan Zhai, and Mi Zhou. Centroid estimation based on symmetric kl divergence for multinomial text classification problem. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1174–1177. IEEE, 2018.

[37] Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

[38] Yu Chen, Lingfei Wu, and Mohammed J Zaki. Graphflow: Exploiting conversation flow with graph neural networks for conversational machine comprehension. *arXiv preprint arXiv:1908.00059*, 2019.

[39] Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. Quac: Question answering in context. *arXiv preprint arXiv:1808.07036*, 2018.

[40] Philipp Christmann, Rishiraj Saha Roy, and Gerhard Weikum. Conversational question answering on heterogeneous sources. *arXiv preprint arXiv:2204.11677*, 2022.

[41] Philipp Christmann, Rishiraj Saha Roy, Abdalghani Abujabal, Jyotsna Singh, and Gerhard Weikum. Look before you hop: Conversational question answering over knowledge graphs using judicious context expansion. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 729–738, 2019.

[42] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[43] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. *arXiv preprint arXiv:1710.10723*, 2017.

[44] Kevin Clark and Christopher D Manning. Improving coreference resolution by learning entity-level distributed representations. *arXiv preprint arXiv:1606.01323*, 2016.

[45] Daniel Cohen, Liu Yang, and W Bruce Croft. Wikipassageqa: A benchmark collection for research on non-factoid answer passage retrieval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1165–1168, 2018.

[46] Gao Cong, Long Wang, Chin-Yew Lin, Young-In Song, and Yueheng Sun. Finding question-answer pairs from online forums. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 467–474, 2008.

[47] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.

[48] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. Overview of the trec 2019 deep learning track. *arXiv preprint arXiv:2003.07820*, 2020.

[49] W Bruce Croft and Roger H Thompson. I3r: A new approach to the design of document retrieval systems. *Journal of the american society for information science*, 38(6):389–404, 1987.

[50] Zhuyun Dai, Vincent Y Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith B Hall, and Ming-Wei Chang. Promptagator: Few-shot dense retrieval from 8 examples. *arXiv preprint arXiv:2209.11755*, 2022.

[51] Jeffrey Dalton, Laura Dietz, and James Allan. Entity query feature expansion using knowledge base links. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 365–374, 2014.

[52] Tivadar Danka and Peter Horvath. modAL: A modular active learning framework for Python. available on arXiv at `https://arxiv.org/abs/1805.00979`.

[53] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. Multi-step retriever-reader interaction for scalable open-domain question answering. *arXiv preprint arXiv:1905.05733*, 2019.

[54] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[55] Bhuwan Dhingra, Kathryn Mazaitis, and William W Cohen. Quasar: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*, 2017.

[56] Ahmed Elgohary, Denis Peskov, and Jordan Boyd-Graber. Can you unpack that? learning to rewrite questions-in-context. *Can You Unpack That? Learning to Rewrite Questions-in-Context*, 2019.

[57] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16, 2006.

[58] Hung-Chieh Fang, Kuo-Han Hung, Chao-Wei Huang, and Yun-Nung Chen. Open-domain conversational question answering with historical answers. *arXiv preprint arXiv:2211.09401*, 2022.

[59] Yair Feldman and Ran El-Yaniv. Multi-hop paragraph retrieval for open-domain question answering. *arXiv preprint arXiv:1906.06606*, 2019.

[60] Song Feng, Kshitij Fadnis, Q Vera Liao, and Luis A Lastras. Doc2dial: a framework for dialogue composition grounded in documents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13604–13605, 2020.

[61] Fernando Ferraretto, Thiago Laitz, Roberto Lotufo, and Rodrigo Nogueira. Exaranker: Explanation-augmented neural ranker. *arXiv preprint arXiv:2301.10521*, 2023.

[62] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2288–2292, 2021.

[63] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. *arXiv preprint arXiv:2212.10496*, 2022.

[64] Sparsh Gupta and Vitor R Carvalho. Faq retrieval using attentive matching. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 929–932, 2019.

[65] Kristian Hammond, Robin Burke, Charles Martin, and Steven Lytinen. Faq finder: a case-based approach to knowledge navigation. In *Proceedings the 11th Conference on Artificial Intelligence for Applications*, pages 80–86. IEEE, 1995.

[66] Stefan Henß, Martin Monperrus, and Mira Mezini. Semi-automatically extracting faqs to improve accessibility of software development knowledge. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 793–803. IEEE, 2012.

[67] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[68] Phu Mon Htut, Samuel R Bowman, and Kyunghyun Cho. Training a ranking function for open-domain question answering. *arXiv preprint arXiv:1804.04264*, 2018.

[69] Hsin-Yuan Huang, Eunsol Choi, and Wen-tau Yih. Flowqa: Grasping flow in history for conversational machine comprehension. *arXiv preprint arXiv:1810.06683*, 2018.

[70] Jizhou Huang, Ming Zhou, and Dan Yang. Extracting chatbot knowledge from online discussion forums. In *IJCAI*, volume 7, pages 423–428, 2007.

[71] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning, 2021.

[72] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

[73] Rolf Jagerman, Honglei Zhuang, Zhen Qin, Xuanhui Wang, and Michael Bendersky. Query expansion by prompting large language models. *arXiv preprint arXiv:2305.03653*, 2023.

[74] Vitor Jeronymo, Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, Roberto Lotufo, Jakub Zavrel, and Rodrigo Nogueira. Inpars-v2: Large language models as efficient dataset generators for information retrieval. *arXiv preprint arXiv:2301.01820*, 2023.

[75] Valentin Jijkoun and Maarten de Rijke. Retrieving answers from frequently asked questions pages on the web. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 76–83, 2005.

[76] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

[77] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[78] Mladen Karan and Jan Šnajder. Faqir–a frequently asked questions retrieval test collection. In *International Conference on Text, Speech, and Dialogue*, pages 74–81. Springer, 2016.

[79] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[80] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[81] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[82] Gary King, Patrick Lam, and Margaret E Roberts. Computer-assisted keyword and document set discovery from unstructured text. *American Journal of Political Science*, 61(4):971–988, 2017.

[83] Sarawoot Kongyoung, Craig Macdonald, and Iadh Ounis. monoqa: Multi-task learning of reranking and answer extraction for open-retrieval conversational question answering. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7207–7218, 2022.

[84] Bernhard Kratzwald and Stefan Feuerriegel. Adaptive document retrieval for deep question answering. *arXiv preprint arXiv:1808.06528*, 2018.

[85] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[86] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[87] Yu-Sheng Lai, Kuao-Ann Fung, and Chung-Hsien Wu. Faq mining via list detection. In *COLING-02: Multilingual Summarization and Question Answering*, 2002.

[88] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[89] Victor Lavrenko and W Bruce Croft. Relevance-based language models. In *ACM SIGIR Forum*, volume 51, pages 260–267. ACM New York, NY, USA, 2017.

[90] Jinhyuk Lee, Seongjun Yun, Hyunjae Kim, Miyoung Ko, and Jaewoo Kang. Ranking paragraphs for improving answer recall in open-domain question answering. *arXiv preprint arXiv:1810.00494*, 2018.

[91] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*, 2019.

[92] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[93] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[94] Yongqi Li, Wenjie Li, and Liqiang Nie. A graph-guided multi-round retrieval method for conversational open-domain question answering. *arXiv preprint arXiv:2104.08443*, 2021.

[95] Yukun Li, Zhenguo Yang, Xu Chen, Huaping Yuan, and Wenyin Liu. A stacking model using url and html features for phishing webpage detection. *Future Generation Computer Systems*, 94:27–39, 2019.

[96] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

[97] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2356–2362, 2021.

[98] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. Distilling dense representations for ranking using tightly-coupled teachers. *arXiv preprint arXiv:2010.11386*, 2020.

[99] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. Contextualized query embeddings for conversational search. *arXiv preprint arXiv:2104.08707*, 2021.

[100] Bang Liu, Mingjun Zhao, Di Niu, Kunfeng Lai, Yancheng He, Haojie Wei, and Yu Xu. Learning to generate questions by learningwhat not to generate. In *The World Wide Web Conference*, pages 1106–1118, 2019.

[101] Yuanhua Lv and ChengXiang Zhai. A comparative study of methods for estimating query language models with pseudo feedback. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1895–1898, 2009.

[102] Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. Zero-shot listwise document reranking with a large language model. *arXiv preprint arXiv:2305.02156*, 2023.

[103] Iain Mackie, Shubham Chatterjee, and Jeffrey Dalton. Generative and pseudo-relevant feedback for sparse, dense and learned sparse retrieval. *arXiv preprint arXiv:2305.07477*, 2023.

[104] Iain Mackie, Shubham Chatterjee, and Jeffrey Dalton. Generative relevance feedback with large language models. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2026–2031, 2023.

[105] Yosi Mass, Boaz Carmeli, Haggai Roitman, and David Konopnicki. Unsupervised faq retrieval with question generation and bert. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 807–812, 2020.

[106] Yossi Matias, Dvir Keysar, Gal Chechik, Ziv Bar-Yossef, and Tomer Shmiel. Generating related questions for search queries, June 13 2017. US Patent 9,679,027.

[107] Jannat Ara Meem, Muhammad Shihab Rashid, Yue Dong, and Vagelis Hristidis. Pat-questions: A self-updating benchmark for present-anchored temporal question-answering. *arXiv preprint arXiv:2402.11034*, 2024.

[108] Edgar Meij, Dolf Trieschnigg, Maarten De Rijke, and Wessel Kraaij. Conceptual language models for domain-specific retrieval. *Information Processing & Management*, 46(4):448–469, 2010.

[109] Ida Mele, Cristina Ioana Muntean, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, and Ophir Frieder. Adaptive utterance rewriting for conversational search. *Information Processing & Management*, 58(6):102682, 2021.

[110] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[111] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[112] Alaa Mohasseb, Mohamed Bader-El-Den, and Mihaela Cocea. Question categorization and classification using grammar based approach. *Information Processing & Management*, 54(6):1228–1243, 2018.

[113] Ali Montazeralghaem and James Allan. Extracting relevant information from user's utterances in conversational search and recommendation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1275–1283, 2022.

[114] Alejandro Moreo, Maria Navarro, Juan L Castro, and Jose Manuel Zurita. A high-performance faq retrieval method using minimal differentiator expressions. *Knowledge-based systems*, 36:9–20, 2012.

[115] Alejandro Moreo, M Romero, JL Castro, and Jose Manuel Zurita. Faqtory: A framework to provide high-quality faq retrieval systems. *Expert Systems with Applications*, 39(14):11525–11534, 2012.

[116] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human generated machine reading comprehension dataset. In *CoCo@ NIPs*, 2016.

[117] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*, 2019.

[118] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Document ranking with a pretrained sequence-to-sequence model. *arXiv preprint arXiv:2003.06713*, 2020.

[119] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. Document expansion by query prediction. *arXiv preprint arXiv:1904.08375*, 2019.

[120] R OpenAI. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2023.

[121] Zhiyuan Peng, Xuyang Wu, and Yi Fang. Soft prompt tuning for augmenting dense retrieval with large language models. *arXiv preprint arXiv:2307.08303*, 2023.

[122] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. Rankvicuna: Zero-shot listwise document reranking with open-source large language models. *arXiv preprint arXiv:2309.15088*, 2023.

[123] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. Rankzephyr: Effective and robust zero-shot listwise reranking is a breeze! *arXiv preprint arXiv:2312.02724*, 2023.

[124] Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, et al. Large language models are effective text rankers with pairwise ranking prompting. *arXiv preprint arXiv:2306.17563*, 2023.

[125] Minghui Qiu, Xinjing Huang, Cen Chen, Feng Ji, Chen Qu, Wei Wei, Jun Huang, and Yin Zhang. Reinforced history backtracking for conversational question answering. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, pages 13718–13726, 2021.

[126] Chen Qu, Liu Yang, Cen Chen, W Bruce Croft, Kalpesh Krishna, and Mohit Iyyer. Weakly-supervised open-retrieval conversational question answering. In *European Conference on Information Retrieval*, pages 529–543. Springer, 2021.

[127] Chen Qu, Liu Yang, Cen Chen, Minghui Qiu, W Bruce Croft, and Mohit Iyyer. Open-retrieval conversational question answering. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 539–548, 2020.

[128] Chen Qu, Liu Yang, Minghui Qiu, W Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. Bert with history answer embedding for conversational question answering. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 1133–1136, 2019.

[129] Chen Qu, Liu Yang, Minghui Qiu, Yongfeng Zhang, Cen Chen, W Bruce Croft, and Mohit Iyyer. Attentive history selection for conversational question answering. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1391–1400, 2019.

[130] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[131] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[132] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[133] Guillem Ramírez, Matthias Lindemann, Alexandra Birch, and Ivan Titov. Cache & distil: Optimising api calls to large language models. *arXiv preprint arXiv:2310.13561*, 2023.

[134] Muhammad Shihab Rashid, Fuad Jamour, and Vagelis Hristidis. Quax: Mining the web for high-utility faq. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1518–1527, 2021.

[135] Muhammad Shihab Rashid, Jannat Ara Meem, Yue Dong, and Vagelis Hristidis. Ecorank: Budget-constrained text re-ranking using large language models. *arXiv preprint arXiv:2402.10866*, 2024.

[136] Muhammad Shihab Rashid, Jannat Ara Meem, and Vagelis Hristidis. Normy: Non-uniform history modeling for open retrieval conversational question answering. *arXiv preprint arXiv:2402.04548*, 2024.

[137] Siva Reddy, Danqi Chen, and Christopher D Manning. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019.

[138] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[139] Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*, pages 232–241. Springer, 1994.

[140] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[141] Joseph John Rocchio Jr. Relevance feedback in information retrieval. *The SMART retrieval system: experiments in automatic document processing*, 1971.

[142] Devendra Singh Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. Improving passage retrieval with zero-shot question generation. *arXiv preprint arXiv:2204.07496*, 2022.

[143] Marzieh Saeidi, Max Bartolo, Patrick Lewis, Sameer Singh, Tim Rocktäschel, Mike Sheldon, Guillaume Bouchard, and Sebastian Riedel. Interpretation of natural language rules in conversational machine reading. *arXiv preprint arXiv:1809.01494*, 2018.

[144] Wataru Sakata, Tomohide Shibata, Ribeka Tanaka, and Sadao Kurohashi. Faq retrieval using query-question similarity and bert-based query-answer relevance. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1113–1116, 2019.

[145] Marija Šakota, Maxime Peyrard, and Robert West. Fly-swat or cannon? cost-effective language model choice via meta-modeling. *arXiv preprint arXiv:2308.06077*, 2023.

[146] Chris Samarinas, Arkin Dharawat, and Hamed Zamani. Revisiting open domain query facet extraction and generation. In *Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 43–50, 2022.

[147] Sheng Shen, Yaliang Li, Nan Du, Xian Wu, Yusheng Xie, Shen Ge, Tao Yang, Kai Wang, Xingzheng Liang, and Wei Fan. On the generation of medical question-answer pairs. In *AAAI*, pages 8822–8829, 2020.

[148] AB Siddique, Fuad Jamour, and Vagelis Hristidis. Linguistically-enriched and context-awarezero-shot slot filling. In *Proceedings of the Web Conference 2021*, pages 3279–3290, 2021.

[149] AB Siddique, Fuad Jamour, Luxun Xu, and Vagelis Hristidis. Generalized zero-shot intent detection via commonsense knowledge. *arXiv preprint arXiv:2102.02925*, 2021.

[150] AB Siddique, Samet Oymak, and Vagelis Hristidis. Unsupervised paraphrasing via deep reinforcement learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1800–1809, 2020.

[151] Eriks Sneiders. Automated faq answering with question-specific knowledge representation for web self-service. In *2009 2nd Conference on Human System Interactions*, pages 298–305. IEEE, 2009.

[152] Hui Su, Xiaoyu Shen, Rongzhi Zhang, Fei Sun, Pengwei Hu, Cheng Niu, and Jie Zhou. Improving multi-turn dialogue modelling with utterance rewriter. *arXiv preprint arXiv:1906.07004*, 2019.

[153] Weiwei Sun, Zheng Chen, Xinyu Ma, Lingyong Yan, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Instruction distillation makes large language models efficient zero-shot rankers. *arXiv preprint arXiv:2311.01555*, 2023.

[154] Weiwei Sun, Lingyong Yan, Xinyu Ma, Pengjie Ren, Dawei Yin, and Zhaochun Ren. Is chatgpt good at search? investigating large language models as re-ranking agent. *arXiv preprint arXiv:2304.09542*, 2023.

[155] Raphael Tang, Xinyu Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Ture. Found in the middle: Permutation self-consistency improves listwise ranking in large language models. *arXiv preprint arXiv:2310.07712*, 2023.

[156] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*, 2021.

[157] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830*, 2016.

[158] Svitlana Vakulenko, Shayne Longpre, Zhucheng Tu, and Raviteja Anantha. Question rewriting for conversational question answering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 355–363, 2021.

[159] Ellen M Voorhees, Dawn M Tice, et al. The trec-8 question answering track evaluation. In *TREC*, volume 1999, page 82. Citeseer, 1999.

[160] Nikos Voskarides, Dan Li, Pengjie Ren, Evangelos Kanoulas, and Maarten de Rijke. Query resolution for conversational search with limited supervision. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 921–930, 2020.

[161] Ben Wang and Aran Komatsuzaki. Gpt-j-6b: A 6 billion parameter autoregressive language model, 2021, 2022.

[162] Liang Wang, Nan Yang, and Furu Wei. Query2doc: Query expansion with large language models. *arXiv preprint arXiv:2303.07678*, 2023.

[163] Shuai Wang, Zhiyuan Chen, Bing Liu, and Sherry Emery. Identifying search keywords for finding relevant social media posts. In *AAAI*, pages 3052–3058, 2016.

[164] Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerry Tesauro, Bowen Zhou, and Jing Jiang. R 3: Reinforced ranker-reader for open-domain question answering. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[165] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. Colbert-prf: Semantic pseudo-relevance feedback for dense passage and document retrieval. *ACM Transactions on the Web*, 17(1):1–39, 2023.

[166] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[167] Thomas Wolf. Neural Coreference - Huggingface. `https://huggingface.co/coref/`.

[168] Zeqiu Wu, Yi Luan, Hannah Rashkin, David Reitter, Hannaneh Hajishirzi, Mari Ostendorf, and Gaurav Singh Tomar. Conqrr: Conversational query rewriting for retrieval with reinforcement learning. *arXiv preprint arXiv:2112.08558*, 2021.

[169] Chenyan Xiong and Jamie Callan. Query expansion with freebase. In *Proceedings of the 2015 international conference on the theory of information retrieval*, pages 111–120, 2015.

[170] Bishan Yang, Jian-Tao Sun, Tengjiao Wang, and Zheng Chen. Effective multi-label active learning for text classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 917–926, 2009.

[171] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv:1902.01718*, 2019.

[172] HongChien Yu, Chenyan Xiong, and Jamie Callan. Improving query representations for dense retrieval with pseudo relevance feedback. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3592–3596, 2021.

[173] Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thoughts representations for cost-efficient reasoning. *arXiv preprint arXiv:2310.03094*, 2023.

[174] Munazza Zaib, Wei Emma Zhang, Quan Z Sheng, Adnan Mahmood, and Yang Zhang. Conversational question answering: A survey. *Knowledge and Information Systems*, pages 1–45, 2022.

[175] Chengxiang Zhai and John Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 403–410, 2001.

[176] Shu Zhang, Dequan Zheng, Xinchen Hu, and Ming Yang. Bidirectional long short-term memory networks for relation classification. In *Proceedings of the 29th Pacific Asia conference on language, information and computation*, pages 73–78, 2015.

[177] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[178] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*, 2023.

[179] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*, pages 207–212, 2016.

[180] Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. Retrieving and reading: A comprehensive survey on open-domain question answering. *arXiv preprint arXiv:2101.00774*, 2021.

[181] Honglei Zhuang, Zhen Qin, Shuguang Han, Xuanhui Wang, Michael Bendersky, and Marc Najork. Ensemble distillation for bert-based ranking models. In *Proceedings of the 2021 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 131–136, 2021.

[182] Honglei Zhuang, Zhen Qin, Kai Hui, Junru Wu, Le Yan, Xuanhui Wang, and Michael Berdersky. Beyond yes and no: Improving zero-shot llm rankers via scoring fine-grained relevance labels. *arXiv preprint arXiv:2310.14122*, 2023.

[183] Shengyao Zhuang, Honglei Zhuang, Bevan Koopman, and Guido Zuccon. A setwise approach for effective and highly efficient zero-shot ranking with large language models. *arXiv preprint arXiv:2310.09497*, 2023.