Retrieval-Augmented Large Language Models

Seminar Thesis of

Sebastian Tewes

KASTEL - Institute of Information Security and Dependability - Karlsruhe Institute of Technology (KIT)
Advisor: M.Sc. Dominik Fuchß

This seminar paper examines the concept of Retrieval-Augmented Large Language Models (RALLMs). This concept merges "Retrieval-Augmented" (RA) with "Large Language Models" (LLM) and represents a new generation of information retrieval (IR) systems. By using the IR systems extended information, the LLMs can also answer questions for which no information was available in the LLMs' training data. This includes proprietary knowledge, but also knowledge that is more recent than the training data. After a brief introduction to the terms information retrieval and large language model, an overview of methods and applications of RALLMs will be given. A particular focus is on what applications there are for RALLMs in software development.

In dieser Seminararbeit wird das Konzept der Retrieval-Augmented Large Language Models (RALLMs) untersucht. Diese Bezeichnung verbindet die Begriffe „Retrieval-Augmented“ (RA) und „Large Language Models“ (LLM) und repräsentiert eine neue Generation von Systemen für den Informationsabruf. Durch die Nutzung der erweiterten Informationen des Retrievalsystems können die LLMs auch Fragen beantworten, zu denen in den Trainingsdaten der LLMs keine Informationen verfügbar waren. Dazu gehört proprietäres Wissen, aber auch Wissen, das aktueller als die Trainingsdaten ist. Nach einer kurzen Einführung in die Begriffe Informationsabruf (Information Retrieval, IR) und großes Sprachmodell (Large Language Model, LLM) wird ein Überblick über Methoden und Anwendungen von RALLMs gegeben. Ein besonderer Fokus liegt darauf, welche Anwendungen es für RALLMs in der Softwareentwicklung gibt.

Contents

1 Introduction

This seminar paper focuses on Retrieval-Augmented Large Language Models (RALLMs), a term that combines "Retrieval-Augmented" and "Large Language Models". This name indicates the beginning of a new era in the field of information retrieval systems. They are traditionally used when searching for information, in chatbots, but also in software development [109].

Information retrieval systems are used as a tool for finding information [109]. The available knowledge should be filtered depending on a user's input, so that only information relevant to the user's input remains. The external data source can be document databases, real-time internet sources, Git repositories or private data. Filtering the information is usually done using sparse vectors / keywords or dense vectors / semantic embeddings.

Large language models (LLMs) model natural language and are able to generate relevant output based on a user's input. If you join the information retrieval system and the large language model, you get a retrieval-augmented large language model [109]. The word "augmented" indicates that the learned weights (parametric memory) of the large language model are enhanced with additional information sourced from the information retrieval system (so-called non-parametric external memory), so that the LLM can also answer questions about this knowledge. There are approaches that fine-tune pre-trained models, train systems end to end and approaches that do not require any training at all as a plug-and-play model.

The RALLM can solve numerous problems faced by LLMs [109]. By using the IR system's extended information, the LLMs can also answer questions for which no information was available in the LLMs' training data. This includes proprietary knowledge, but also knowledge that is more current than the training data. In addition, LLM hallucinations and factual inaccuracy can be reduced and LLM responses become more reliable and precise.

This seminar paper is structured as follows: It begins with a brief discussion of the terms "Information Retrieval" (IR) and "Large Language Models" (LLMs) in section 2. What follows is an overview of the methods of the Retrieval-Augmented Large Language Model (RALLM) pipeline in section 3 with a particular focus on query rewriting, retrieval, reranking and readers. Finally, applications for RALLMs will be presented in section 4. General applications for RALLMs will first be presented, such as question answering. Then a special focus will be placed on which applications RALLMs are used and how they are used in software development, such as code generation. It should also be noted that this is neither a paper on LLMs nor a paper on processes in software development. The terms from these subject areas that are necessary for further understanding are briefly explained.

The topic of RALLMs has experienced a major upswing since the breakthrough of large language models in the early 2020s and is undergoing dynamic change. This means that a standard process has not yet been established for many challenges related to RALLMs. Therefore, many of the scientific publications break new ground. More than 50% of the papers cited in this seminar paper are from 2023 or newer, so that a review process has not (yet) taken place for some papers.

2 Background

This section discusses information retrieval and large language models.

2.1 Information Retrieval

According to section 1, information retrieval systems are used as a tool for finding information. The term information retrieval (IR) has undergone an evolution since its creation in the 1950s [80]. The new ability of computers to store large amounts of data made it necessary to mine this data for useful information. When the National Institute of Standards and Technology (NIST) launched the Text Retrieval Conference (TREC) [26] in 1992, the field of research experienced a boom [80]. With search engines like Google [6], algorithms were developed for the first time that could search the Internet.

Due to progress in computational linguistics, the focus of the research field has shifted to interaction with natural language with so-called user queries (also called user prompts or user inputs) [109]. However, the core functionality of the IR system has remained the same: filtering information from a (large) defined context of knowledge that is relevant to the user. The approach to information retrieval has evolved over time. Previous approaches filter documents based on sparse vectors (keyword matching) [69]. More recent approaches are based on dense vectors [36], which determines the semantic similarity of text sections.

2.2 Large Language Models

Large language models (LLMs) model the generative likelihood of word sequences and they predict the probability of the following words, so that they can generate text in a natural language [106]. LLMs are the result of an evolutionary process in Natural Language Processing (NLP). There are the following stages of development (see Figure 1): statistical language models (SLM) based on the markov assumption, neural language models (NLM) based on multi-layer perceptron (MLP) and recurrent neural networks (RNNs), pre-trained language models (PLM) based on bidirectional long short-term memory (biLSTM) or Transformer architecture [88] with the pre-training and fine-tuning learning paradigm, large language models (LLM) as scaling PLM with billions of parameters.

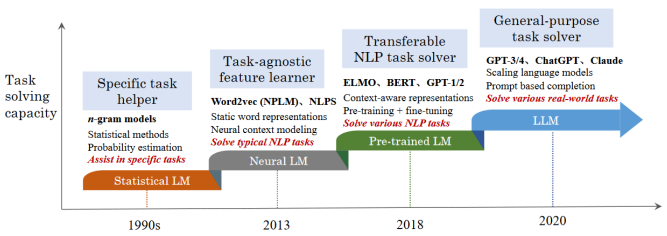
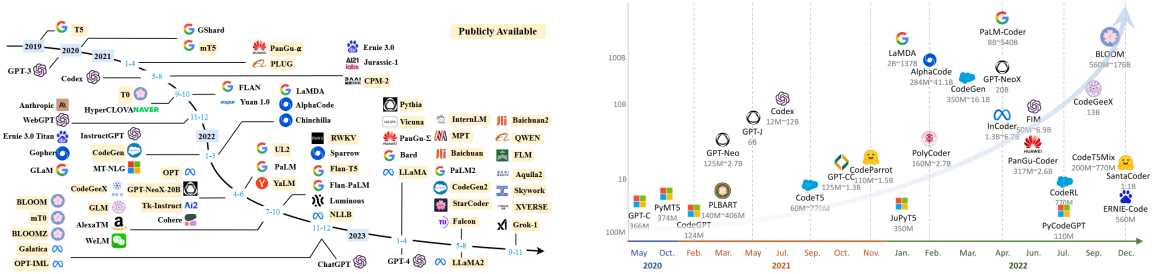


Figure 1: paradigms for language models [106]

As sequence-to-sequence models (seq2seq), LLMs transform an input sequence into an output sequence. Interaction with users usually takes place on a text basis. Modern models are based on the transformer architecture [88] with an attention mechanism. The transformer architecture is based on the encoder-decoder structure, where the encoder maps an input sequence x in a continuous representation z. The decoder maps this representation z to an output sequence y. The attention mechanism maps a query and a key-value pair to an output, where the output is a sum of the values weighted



(a) LLM timeline [106] (b) NL2Code models timeline [101]

Figure 2: LLM examples

depending on the query and key, i.e. the more relevant a key is to a query, the more its value is weighted. Important information is emphasized, irrelevant information is suppressed.

According to Liu et al. [50] and Zhao et al. [106], different paradigms have emerged in the use of LLMs. In the pre-train and fine-tune paradigm, a model with a fixed architecture is pre-trained on large data sets in order to learn general-purpose features. The pre-trained model is then adapted to downstream tasks with smaller task-specific data sets. In the "pre-train, prompt, and predict" paradigm [50], downstream tasks are reformulated using textual prompts so that they are comparable to the pre-training tasks. Additional task-specific training can often be avoided, making a single general-purpose LLM suitable for numerous tasks, provided the LLM receives appropriate prompts.

There are numerous LLMs in the literature and in practice (see Figure 2a). Some of them will be briefly presented here. BERT (Bidirectional Encoder Representations from Transformers) [16] is a language model developed by Google AI based on the Transformer architecture with 110M total parameters. It can process text bidirectionally, which is a significant improvement over previous models that process text sequentially. It is a pre-trained model that can be fine-tuned with an additional output layer. BART [43] is a denoising autoencoder from Facebook AI for pre-training sequence-to-sequence (seq2seq) models. It uses the Transformer architecture and represents a generalization of BERT. It is particularly suitable for fine-tuning. GPT-3 [7] from OpenAI (further development GPT-4 [61]) is an up-scaled LLM with 175 billion parameters (10x more than any previous non-sparse language model) with strong task-agnostic performance without the need for fine-tuning. LLaMA [86] and the further development Llama 2 [85] from Meta represent a collection of open source models with up to 70B parameters.

All of these LLMs are also suitable for generating programming code described using natural language (NL2Code). However, there are also some models that are specialized for these NL2Code tasks (see Figure 2b). A milestone is Codex [11], a GPT-based model that has been fine-tuned on GitHub Python code and is the basis for GitHub Copilot[1]. Zan et al. [101] provides an overview of 27 LLMs that specialize in the generation of programming code, for example CodeT5+ [95] (see section 4).

[1] https://github.com/features/copilot

3 Retrieval-Augmented Large Language Models: Methods

A Retrieval-Augmented Large Language Model (RALLM) represents, as described in section 1, the combination of an information retrieval system and a large language model. This section describes the methods in more detail and explains how these parts are combined and assembled into a type of pipeline.

Based on Zhu et al. [109], a pipeline consisting of the four modules (see Figure 3) query rewriter, retriever, reranker and reader has proven to be a standard process for RALLMs. The process begins with a user question (hereinafter called query), which can optionally be adapted by the query rewriter module. The retriever module then searches the available knowledge for documents or text passages that are relevant to the query. The optional reranker module can look at these documents or text passages in more granular detail. Finally, the text passages found in the previous modules are presented to a reader module, usually consisting of an LLM, together with the user question in order to answer the user query. There are many variants of these modules, and with this modular approach many variants can be combined with each other as desired. A retriever and a reader must be present in every RALLM, query rewriters and rerankers are optional.

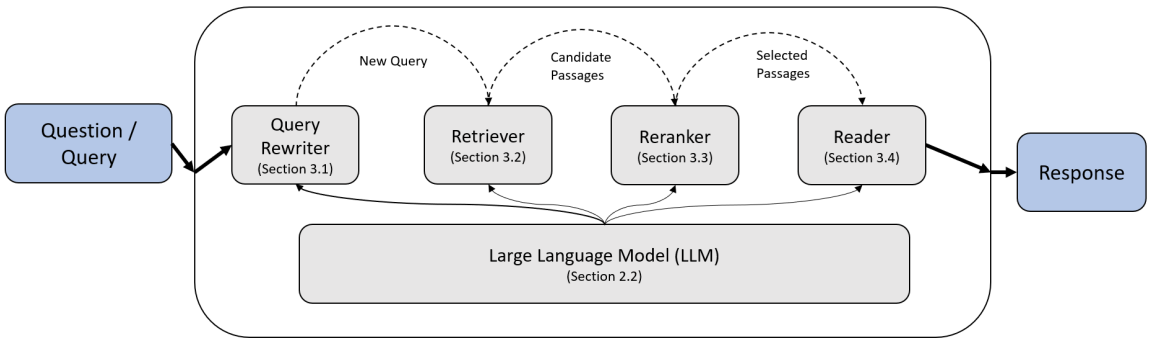


Figure 3: RALLM Pipeline, adapted from Zhu et al. [109]

The set of knowledge available to the RALLM (often called document repository or knowledge base) needs to be clearly defined, and its accessibility must be ensured. This can be closed knowledge (set of text files) or open knowledge (e.g. the Internet). The knowledge base can change dynamically without any training.

In the following sections (3.1 - 3.4) the four modules of the described pipeline process for RALLMs, namely the query rewriter, retriever, reranker, and reader, will be examined in more detail along with their different details and characteristics. These approaches are based on so-called in-context learning [7, 17, 68], in which an LLM is intended to answer a user query based on a provided context and not on the previously learned knowledge. However, there are also some approaches for RALLMs that cannot be classified into this standard pipeline. These should be discussed in the section 3.5. Finally, a brief overview of evaluation methods (section 3.6) for RALLMs will be given.

3.1 Query Rewriter

The "Query Rewriter" module of a RALLM is intended, according to Zhu et al. [109], to increase the precision and meaningfulness of a user query at the beginning of the pipeline. There are approaches to reformulate user queries (rewriting, expansion, multi-queries) or approaches to generate a pseudo-answer for the retriever step.

Query rewriting approaches respond to the fact that original user query are often short, contradictory or imprecise [109]. The rewriting approaches are intended to give the retriever the opportunity to find exactly the documents that are relevant to the user. They improve the results of the RALLM, but are not mandatory.

A common option for query rewriting is a query expansion [8], which adds additional information to a query. Here an LLM is given an original query q and is asked to reformulate it and provide different variants of the query q' with similar terms [15]. A vocabulary mismatch, especially in sparse retrievers, can be avoided because the extended query can recognize relevant documents that have no lexical and only a slight semantic match with the original query. The original query is then usually concatenated with the generated variants of the query and passed to a retriever [32].

Ye et al. [99] describes four properties that a rewritten query should have: Correctness (preserve the meaning of the original query), Clarity (unambiguous and independent of the conversational context), Informativeness (incorporate valuable, relevant and useful information from the conversational context), Non-redundancy (avoid duplicating context). Experimental results show that query expansion approaches are particularly suitable for sparse retriever approaches and then significantly increase the performance of a RALLM.

Hypothetical Document Embeddings (HyDE) [22] lets an LLM generate a hypothetical pseudo document based on a user query (see Figure 4). With the following (dense) retriever module relevant documents with semantic similarity to the generated hypothetical document are searched. A semantic similarity between the original user query and the document corpus is no longer determined. This reduces the number of noisy documents and eliminates the need for many retrievers to assume that questions and answers have a high semantic similarity.

Generate-and-Retrieval (GandR) [103] is an approach comparable to HyDE that carries out the retrieval process with a preliminary prediction as a pseudo answer. The retrieval process is then carried out with this psedo-response. In contrast to HyDE, it requires training and it is embedded in a generate-and-retrieval framework.

HyDE and GandR are particularly suitable for dense retriever approaches, outperforming numerous classic dense retriever approaches. However, they require more computing time than conventional approaches. Query2doc [92] is an approach that extends the query for sparse and dense retrievers with generated pseudo-documents and, for example, outperforms approaches such as HyDE.

In the information retrieval paradigm conversational search [66], an (iterative) interaction takes place between the user and the retrieval system. Query rewriting approaches also exist for conversational search, for example from Mao et al. [55] and Ye et al. [99].

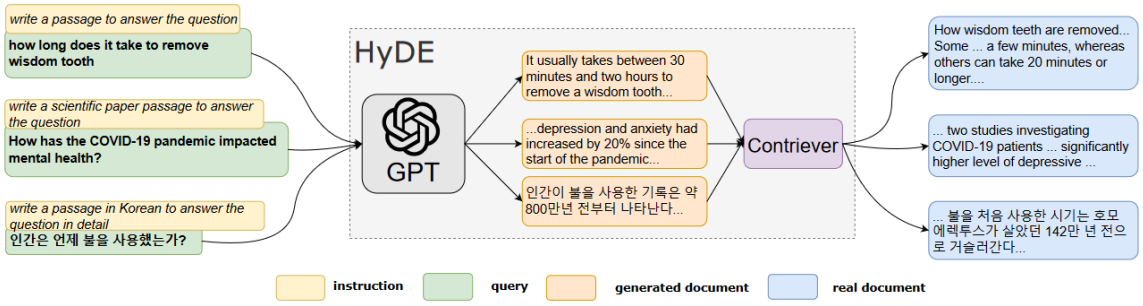


Figure 4: HyDE [22]

Ma et al. [54] represents a fundamentally different query rewriting approach compared to previous approaches. An end-to-end rewrite-retrieve-read pipeline trained with reinforcement learning is proposed, i.e. the rewriter is supported by the reader feedback trained. Compared to trained rewriters and frozen (blackbox) rewriters, the end-to-end reinforcement learning approach shows significant performance gains on numerous data sets. However, the end-to-end training means it deviates significantly from the previously adopted modular approach of the RALLM pipeline.

3.2 Retriever

The "Retriever" module of a RALLM is, according to Zhu et al. [109] and Guo et al. [24], a first-pass document filter that has the task of delivering relevant documents or text passages based on the user query from the available knowledge. This returns a subset of the document repository/corpus that is relevant to the user query (or the output of the query rewriting module). Expressed formally [107]: Let q be a natural language query and d_i be a text or document from a large document repository $D = \{d_i\}_{i=1}^{m}$ consisting of m documents. Now a ranked list $\mathcal{L} = [d_1, d_2, ...d_k]$ of the k « m most relevant documents should be returned based on a relevance score of the retrieval model. If a reranking mechanism is used in the RALLM pipeline, an initial set of k document candidates is searched with the aim of achieving a high recall. The reranking module then takes over the more fine-grained consideration with the aim of good precision (see Figure 5).

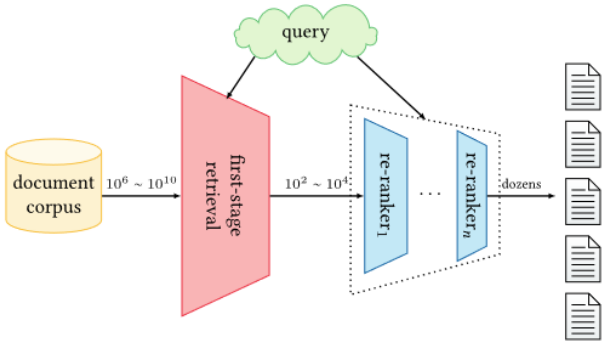


Figure 5: Multi-Stage Architecture [24]

According to Guo et al. [24], there are two main paradigms for retrievers: classical term-based models (sparse retriever) and semantic models (dense retriever) with word embeddings. Formally speaking, most or all elements of a dense vector contain non-zero values. In sparse vectors, most of the elements are zero and the few non-zero values carry the essential information. In addition, there are hybrid approaches (Table 1 in the survey

from Guo et al. [24] offers a list of methods).

Sparse retrieval methods [24] usually represent each document and each query as a sparse vector, usually based on a discrete symbolic representation (e.g. bag of words). Sparse retrieval methods can be further divided into two classes (neural weighting schemes, sparse representation learning), depending on how the sparse representation is learned. A key advantage of the sparse retriever is its clarity and interpretation options. Since no semantic information is taken into account, the so-called vocabulary mismatch problem (no consideration of similar words) and the lack of consideration of word order are disadvantages of sparse retrievers.

BM25 [69] is a popular traditional sparse retriever approach based on a Probabilistic Relevance Framework (PRF). It is based on the term frequency (TF) in the document and their inverse document frequency (IDF), where common terms are considered less important and rare words are given higher weight. The term frequency is normalized with the document length to prevent longer documents from being systematically preferred. In addition, various models and variations derived from BM25 exist.

SPLADE [21] is a newer approach to a sparse retriever. It uses a weighted representation of words that takes into account not only frequency but also semantic meaning based on BERT [16]. Experiments show that it can keep up with dense retrieval methods, but at the same time requires fewer computing resources and is more efficient.

Dense retrieval methods [24] use word embedding techniques and continuous vectors taking into account word semantics instead of a discrete symbolic representation. Word embedding techniques [57] learn word representations using neural networks (here continuous skip-gram model) and thus map linguistic rules and patterns in the learned vectors. They learn precise syntactic and semantic word relationships so that they can be 'calculated' with, for example: $vec('Madrid') - vec('Spain') + vec('France') \approx vec('Paris')$. By taking word semantics into account, word embedding techniques solve the vocabulary mismatch problem of sparse retrievers.

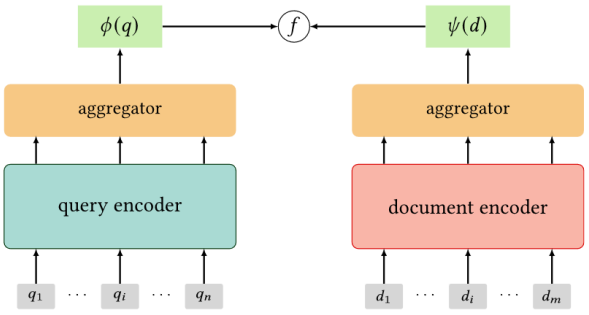


Figure 6: Dense Retrieval Architecture [24]

Dense retrieval methods can also be further divided [24] into two classes (term-level representation learning and document-level representation learning), depending on whether the vectors are based on individual sentences/sequences (fine-granular) or based on entire documents (coarse-granular) are formed.

In order to find suitable documents d for a query q, dense vectors are determined for the query q and for all available documents d (see dual-encoder architecture or siamese network in Figure 6), i.e. there are independent dense vectors for the queries $\phi(q)$ and the documents $\psi(d)$. In a matching layer f, a final relevance score is calculated from the

learned representations $\phi(q)$ and $\psi(d)$, often using a similarity function. The k most relevant documents according to this score are selected.

Zhao et al. [107] offers a further overview of dense text retrieval methods, based on over 300 papers. In particular, the aspects of architecture, i.e. how to design the dense retrieval architecture, training, i.e. how to optimize the training of the dense fetcher, indexing, i.e. how to design efficient data structures for indexing and retrieving dense vectors, and integration, i.e. how to integrate a complete retrieval pipeline.

Dense Passage Retrieval (DPR) [36] is a common approach for a dense retriever. Because it is used in many applications (see Section 4.1), it is discussed in more detail here. DPR do not form vector representations at the level of the entire document but at the level of passages in the document, i.e. they split each document d into passages p as a sequence of tokens w. Each passage is approximately the same length; The ideal length of a passage, often called chunk, and its separations is a function of the retriever and the reader and is further investigated by Wang et al. [96]. The embedding model is trained so that the inner product between the question and the relevant passage at the vector level is maximized for a batch. DPR further uses a dense encoder $E_P(\cdot)$, which maps each of the text passages onto a d-dimensional vector. At runtime, the user query is also mapped onto a d-dimensional vector using an encoder $E_Q(\cdot)$ and the k passages with the shortest distance to the query vector are selected, called maximum inner product search (MIPS). The distance or similarity between the query and the text passage is determined by the dot product of their vectors: $sim(q, p) = E_Q(q)^T E_P(p)$. If the vectors are normalized to 1 (unit vector), the cosine similarity is equivalent to the inner product. Alternatively, the euclidean distance (L2) can also be used.

In addition to DPR, there are other approaches for embeddings and dense retrievers.

A common and commercial model for text embeddings is text-embedding-ada-002[2] from OpenAI, provided via an API endpoint, with a maximum length for input tokens of 8191 and a fixed vector output dimension of 1536.

FAISS [34] is an open source library for efficient similarity search in dense vectors. Various approaches to how the distance or similarity between dense vectors can be determined are discussed. In addition, high-performance and parallel implementations on GPUs are proposed.

Adapted Dense Retrieval (Adder) [1] (under review) represents an extension of the Dense Retrieval method. Embeddings are transformed task-specifically, so that the retrieval result improves for small k (k=1 or 3 or 5).

Contriever [31] represents a dense retriever that was trained without supervision and still achieves good performance in various retrieval settings, with reranking (see Section 3.3) it even outperforms DPR [36]. This enables the development of trained domain-specific retrievers without the need for labeled data sets. Based on contrastive learning, an auxiliary task, the Inverse cloze task (ICT), takes over the supervision by predicting the surrounding context given a query.

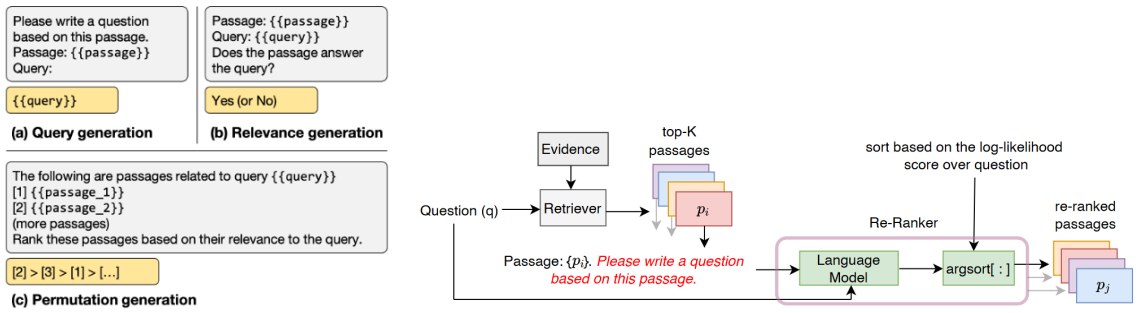
[2] https://platform.openai.com/docs/guides/embeddings

3.3 Reranker

The "Reranker" module of a RALLM is, according to Zhu et al. [109], a second-pass document filter that has the task of evaluating or ranking a list of documents or text passages in terms of relevance to the user query. Such a ranking is more fine-grained than the previous retriever step and it focuses more on the quality of the documents, but is not mandatory. Due to the lower but higher quality context, performance and latency also improves. The reranker module assumes that the Retriever module provides a list of k documents or text passages. There are essentially two approaches to reranking: fine-tuning a specific reranking model / LLM or prompting an LLM for reranking.

Fine-tuned reranking models evaluate the documents or text passages found in the retriever module for relevance to the user query. Reranking is defined as a classification task.

MonoT5 [60] uses a generation model and trains it to return the tokens 'true' or 'false', depending on the relevance of document d with respect to query q. Depending on the estimated probability of relevance P(relevant=1|q,d), determined exclusively by softmax to 'true' or 'false', the document chunks can then be sorted. The paper also shows how such a ranking model can be trained and that a sequence-to-sequence architecture outperforms an encoder-only architecture. RankT5 [110] refers to this and shows that a direct numerical determination of the ranking scores instead of generating text tokens outperforms the approach of Nogueira et al. [60] by 2%.



(a) Reranking Prompt Types [83] (b) UPR Reranking Scheme [72]

Figure 7: Reranking

As an alternative to the first approach, LLMs can be used to rerank documents or text passages through prompting. There are different prompting approaches (see Figure 7a) [83]: The relevance generation approach asks the LLM whether an individual document (pointwise) appears relevant to a query. The query generation approach asks the LLM for a query for each document (pointwise), whereby the relevance of each document depends on the logarithmic probability of generating the respective query given this document. The permutation generation approach prompts the query and a list of documents (listwise, often with sliding windows due to the long context) into an LLM and asks it to order the documents according to their relevance to the query.

The Unsupervised Passage Reranker (UPR) [72] uses a pretrained zero-shot question generation model to rescore document passages. It can be applied on top to any retriever and does not require any task-specific fine-tuning due to the unsupervised approach. It rescores each of the k document passages p_i from the retrieval step by calculating the likelihood of the input question q given the document passage p_i (see Figure 7b). Experiments show that the reranking of the top 1000 document chunks of the contriever [31] (see Section 3.2) in the top-20 retrieval accuracy metric outperforms the DPR [36].

However, UPR [72] relies on the availability of the model's logarithmic output, which is not the case for many commercial models. This problem is addressed by various works. The RankGPT approach [83] responds to this by proposing an alternative permutation generation approach, whereby the LLM (here GPT-4) should suggest permutations of a group of passages. A sliding window approach responds to the limited context window. This approach outperforms previous approaches. Since this method relies on proprietary models behind API endpoints, RankVicuna [64] was developed as an open-source reranking model with a zero-shot setting whose performance lies between GPT-3.5 and GPT-4. Cho et al. [13] further investigates how prompts can be optimized for zero-shot reranking models.

3.4 Reader

The "Reader" module of a RALLM, according to Zhu et al. [109], has the task of generating an answer based on the user query and taking into account the received text passages or documents and presenting it to the user. This reader is usually represented by an LLM (see section 2.2). A distinction is made between passive and active readers.

A passive reader uses the received documents or text passages directly and presents them to the LLM, whereby the IR system and the LLM are basically independent [109].

REPLUG [77] (under review) views the LLM in a RALLM as a blackbox, in contrast to many previous approaches that fine-tune LLMs. The retriever (and possibly reranker) is adapted to the LLM and the documents received are inserted together with the query into the input prompt of the LLM, which then provides the final answer to the query. This also enables the use of commercial blackbox APIs, which - without being able to see parameters or internal information of the model - send an output back to an input. The Retriever component of REPLUG is potentially tunable.

The In-Context RALM [68] is an approach that leaves the architecture of the (L)LM unchanged and requires no further training or fine-tuning. In contrast to REPLUG [77], which only requires one retriever operation, the in-context RALM [68] requires a retrieval operation every s (retrieval stride) tokens. Experiments show that a higher frequency of retriever operations leads to improved performance, but at the expense of a significantly longer runtime. Therefore, Ram et al. [68] propose to find a balance between performance and runtime by retrieving every s=4 tokens.

In contrast to passive readers, an active reader can trigger the retrieval pipeline if they think it is necessary [109]. Here the IR system and LLM are usually not independent, so this paradigm also represents a small deviation from the standard RALLM pipeline.

Forward-Looking Active REtrieval augmented generation (FLARE) [33] is a method that actively decides when and what to retrieve. It iteratively generates a temporary next sentence. If this contains low-probability tokens, the generated temporary sentence is used as a query to retrieve documents and then a final sentence is generated. Demonstrate-Search-Predict (DSP) [37] is a high-level program that can systematically break down problems into small problems while bootstrapping pipelines. Composable functions bootstrap training examples (Demonstrate), information from a knowledge corpus is recorded (Search) and an output is generated (Predict).

Readers (active and passive readers) can be expanded through the Chain of Thought (CoT) paradigm. CoT [97] is a paradigm with a series of intermediate reasoning steps that extends a prompt as a classic input-output pair to <input, chain of thought, output>.

Multi-Chain Reasoning (MCR) [100] prompts an LLM with multiple chains of thought. Information and facts are then selected from several reasoning chains to generate an answer. A decomposition model and a retriever iteratively generate a reasoning chain, which are then merged into a multi-chain context, which is then passed along with the original question to the meta-reasoner model.

3.5 Further RALLM Models and additional Aspects

This section presents some approaches to RALLMs that deviate from the previous pipeline and also discusses other aspects.

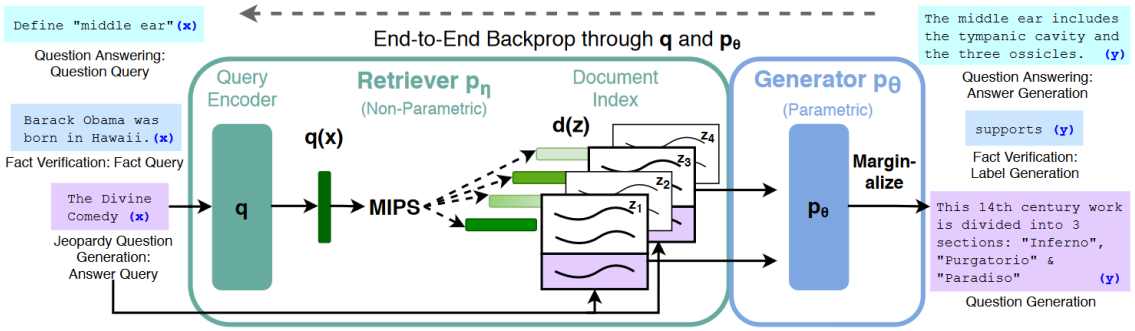


Figure 8: RAG training (end-to-end) [44]

Lewis et al. [44] presents a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG). Similar to the RALLM pipeline presented previously, a pre-trained sequence-to-sequence (seq2seq) model is combined as parametric memory with a dense vector as non-parametric memory. In contrast to the previously presented RALLM pipeline, the model is trained end to end with back-propagation (backprop [70]) and query input x is mapped directly to the generated answer y as output using the retrieved text documents z (see Figure 8). Extensive training of the model is necessary and the components used (retriever, generator) are trained together and are no longer independent of each other. The retrieval component p_η is based on the DPR [36] (see Section 3.2) and the generator component p_θ is based on BART [43] (see Section 2.2). The top k documents are found via

a Maximum Inner Product Search (MIPS), whereby the query x is translated with q(x) and the documents z with d(z) into a dense vector representation. More generally, RAG is a method that aims to link retrieval and generation components end-to-end in one model. RALLMs only aim to extend an LLM with a retriever, so the RAG is a specific RALLM implementation.

Sachan et al. [71] represents another approach to training a RALLM end to end. Unsupervised pre-training is followed by supervised fine tuning. Through end-to-end training, better results in retrieval accuracy and answer extraction can be achieved; this requires resource-intensive training and flexible modularization is not possible.

Autoencoding-based Retriever Training (ART) [73] introduces a new autoencoding training scheme for dense retrieval models that only requires unpaired inputs and outputs and no longer requires labeled training data. The retrieved documents are viewed as a noisy representation of the query, with the question reconstruction probability providing soft labels for the relevance of the document.

The sequential nature of the retrieval augmentation pipeline increases the waiting time (latency) for an answer compared to classic search algorithms or simple LLM queries [82]. This is because the LLM often needs to be given long contexts as input and the computational complexity in a self-attention transformer layer increases quadratically with the input sequence [88]. Additionally, running LLMs with billions of parameters is often not possible on small computing environments. To improve response times, it is advisable to transfer compression techniques for neural networks such as Knowledge Distilling [28] to RALLM approaches. Distillation generally involves transferring knowledge from a large, slow model to a smaller, fast model. QUILL [82] and ReAugKD [105] are respectively approaches that apply knowledge distillation approaches to RALLMs through a professor-teacher-student approach. For example, ReAugKD achieves state-of-the-art results on the GLUE benchmark with less than 3% latency overhead compared to the baseline without retrieval augmentation.

The RALLM approaches considered in this seminar paper so far are limited to text. MuRAG [12] and RA-CM3 [98] are models that extend RALLMs multimodally so that they (retriever, generator/reader) can process images in addition to text.

3.6 Evaluation

There are numerous approaches to evaluate the performance of a RALLM methods. According to Section 1, a RALLM using external knowledge is intended to make an LLM more precise and reliable, while reducing the problem of hallucinations and outdated and unknown knowledge.

According to Es et al. [19], there are essentially two dimensions to consider when evaluating a RALLM: the ability of the retriever to obtain relevant documents and the ability of the LLM/reader to generate an answer to the question using the information from the retriever.

Chen et al. [10] describes that a RALLM should have four abilities: noise robustness, negative rejection, information integration, and counterfactual robustness. Noise Robustness is the ability of the RALLM to extract the relevant information from a noisy document. Negative rejection is the ability of the RALLM to reject an answer to a question if the required information is not present in the (received) documents. Information integration is the ability of RALLM to answer complex questions that require combining information from multiple documents to answer them. Counterfactual robustness is the ability of the RALLM to detect factual errors in the documents. Figure 9 shows a concrete example for each of the four abilities. Data sets have been created for all four capabilities of the RALLM [10].

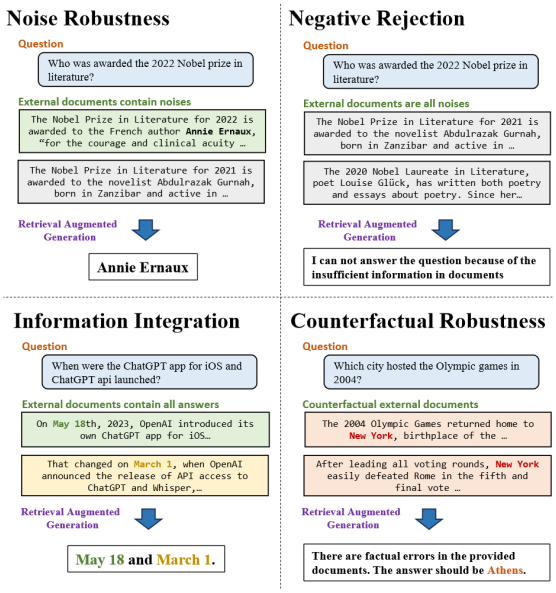


Figure 9: RALLM Abilities [10]

Retrieval Augmented Generation Assessment (RAGAS) [19] is a framework for reference-free evaluation of Retrieval Augmented Generation (RAG) pipelines. It provides metrics (including faithfulness, answer relevance, context relevance) to evaluate a RALLM even without human annotations, allowing LLMs to evaluate the relevance of a document or the quality of a question through prompting.

4 Retrieval-Augmented Large Language Models: Applications

In this section, various applications for RALLMs will be presented. General applications, especially question-answering systems, including those with private knowledge, and web browsing systems will be presented in Section 4.1. RALLM applications in software engineering will then be presented in Section 4.2. The focus here is on applications that use RALLM approaches. Further LLM based approaches for these applications are covered by Kaddour et al. [35].

4.1 General Applications

This section presents general applications for RALLMs. These are primarily question answering systems that are expanded with general publicly available data sources such as Wikipedia or the Internet, special public data sources such as in medicine or private data sources.

LLMs give an answer to a question as input as an output in natural language and thus represent a question answering system, thus extending classic open domain question

answering systems (ODQA). ODQA's task is to answer factoid questions correctly. The first major evaluation of domain-independent question answering systems took place back in 2000 at TREC-8 (see Section 2.1) [90]. Fact-based questions should be answered from approximately 1.5 gigabytes of text, with a ranked list of 5 pairs [document-id, answer-string] being returned for each question. Since then, the ODQA task has been improved and various benchmarks have been created, such as SQuAD [67], Natural Questions (NQ) [40] and WebQuestions (WQ) [5].

The Dense Passage Retriever (DPR) [36] presented in Section 3.2 was developed for ODQA. Experiments show that DPR performs better than the sparse retriever BM25 [69] (see Section 3.2) on the data sets NQ and WQ.

RAG-end2end [81] extends the RAG approach [44], which was presented in Section 3.5. Through the extension, this can be adapted to a domain-specific knowledge base, whereby the retriever component and the generator component are trained together end-to-end (instead of a separate fine-tuning of the DPR). The evaluation on data sets from the COVID-19, conversations and news domain shows that end-to-end training leads to better results.

The Hybrid Hierarchical Retriever (HHR) [2] combines sparse retrieval methods and dense retrieval methods in a hierarchical structure (document retrieval followed by a passage retriever, see Figure 10), similar to the retrieve-reranking pipeline. Hierarchical, because first top-k_d documents are retrieved, from which the top-k_p passages are then retrieved. For both steps, sparse or dense retriever can be selected or the hybrid variant, resulting in a total of 9 possible configurations. This shows that dense and sparse retrievers can complement each other well.

In addition to the ODQA data sets mentioned, it is common for RALLM applications to use a Wikipedia dump[3] as the (only) source of knowledge (see Figure 10). Wikipedia contains up-to-date knowledge in natural language. The machine reading at scale (MRS) setting [9] first uses the document (sparse) retriever to find 5 relevant articles from the over 5 million english-language wikipedia articles, which are then retrieved by the document reader be examined in more detail. It can be shown that the 2017 approach outperforms the classic built-in Wikipedia search engine[4] (e.g. 77.8% vs. 62.7% on SQuAD). However, the MRS approach is generic and not just limited to Wikipedia.

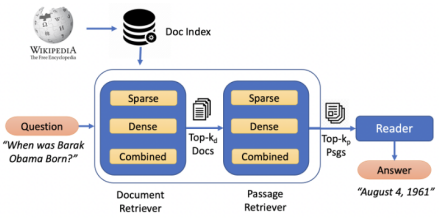


Figure 10: HHR with Wikipedia [2]

Since ODQA is often ambiguous and often allows multiple interpretations of the question-answer combination, it often makes sense to generate a longer answer while taking into account multiple possible interpretations. The Tree of Classifications (TOC) [38] framework takes this challenge into account by recursively constructing a tree of disambiguations for ambiguous questions. This results in improvements in performance.

[3] https://dumps.wikimedia.org/enwiki/latest/
[4] https://www.mediawiki.org/wiki/API:Search

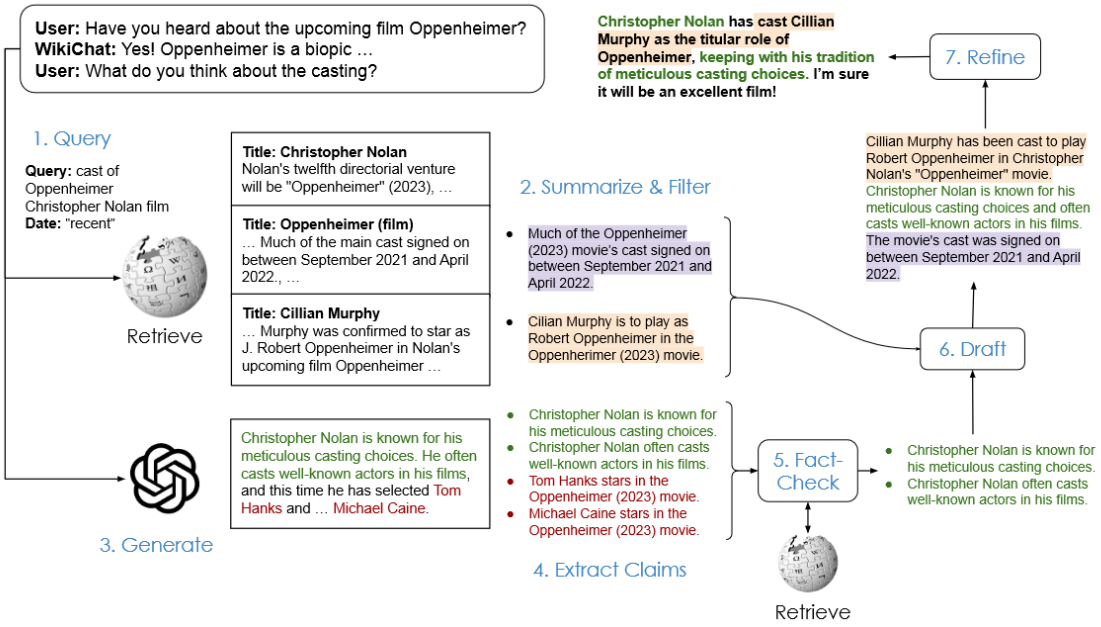


Figure 11: WikiChat [75]

WikiChat [75], based on GPT-4, represents a very new, high-performance approach to integrating information from Wikipedia into an LLM chat. It is optimized for the three metrics factuality, conversationality and latency. WikiChat achieves 97.3% factual accuracy in conversations, in contrast to GPT-4's 66.1%. That's why they call it the first LLM-based few-shot chatbot that almost never hallucinates and has high conversational ability and low latency. The 7-stage pipeline (see Figure 11) combines the best of all information retrieval approaches. Stage 1 generates a query that is sent to an information retrieval system and returns N_{IR} passages. Stage 2 extracts relevant parts of the preserved passages and summarizes them in bullet points. Stage 3 generates a response for the conversation history. Stage 4 divides the answer into multiple claims. Stage 5 verifies each claim using chain-of-thought prompting. Stage 6 generates another draft answer based on the bullet points and the verified claims. Stage 7 refines the answer based on relevance, naturalness, non-repetition and temporal accuracy and provides a final answer. However, due to numerous API calls, latency worsens and costs are higher compared to GPT-4. Therefore, WikiChat provides a second, smaller model with knowledge distillation (see Section 3.5) and still achieves a factual accuracy of 91.1% with a latency comparable to GPT-4. A real user study with 40 participants confirms the better results in terms of factual accuracy of WikiChat compared to GPT-4.

RALLMs can also be used to verify facts and statements in order to find fake news (see FakeNews Challenge[5]). Fact Extraction and VERification (FEVER) [84] is a data set of 185,445 claims, which are divided into 3 categories: Supported, Refuted and NotEnoughInfo. A baseline is provided, with the retrieval component being implemented using a

[5] http://www.fakenewschallenge.org/

sparse retriever. Guzman Olivares et al. [25] will set a new state of the art on the FEVER benchmark for fact verification in 2023, based on an approach that takes Wikipedia's graph structure into account.

The RALLM applications for ODQA or fact verification presented so far were limited to a specific data set or Wikipedia as an external knowledge repository. As an extension to this, RALLMs were developed that can access the Internet via a web search and use the knowledge. WebGPT [59] is an extension of GPT-3 developed by OpenAI, where the LLM searches the Internet through the Microsoft Bing Web Search API[6] and can answer a user's question based on the information found there. Lazaridou et al. [42] (under review) publish a comparable work that relies on the Google Search Engine.

The RALLM applications presented so far were designed for open-domain knowledge and were not limited to a specific area of knowledge. However, there are also approaches that are limited to a specific area and are more effective in this area.

Mavi et al. [56] applies existing question answering systems to specific domains such as law and finance using chain-of-thought prompting. It should be noted that data in these domains is often only semi-structured and often contains numbers in table format and therefore requires special treatment during retrieval.

PaperQA [41] (under review) applies the RAG model [44] (see Section 3.5) to question answering tasks for scientific literature. It provides information retrieval on the entire scientific article and can provide answers to questions. For this purpose, a benchmark data set for RAG for scientific papers LitQA was created for further improvements in the future.

Chat-Orthopedist [78] represents a shared decision making tool for adolescent idiopathic scoliosis (AIS, spinal deformity) patients, which is based on a RALLM (see Figure 12). It consists of three components: an external AIS knowledge base (document chunks with 2000 tokens), a retriever (dense retriever with embedding size d=768) and an LLM (ChatGPT). Chat-Orthopedist marks a significant advancement beyond earlier models, offering numerous benefits. It leverages a database with options for rapid updates and ensures source transparency. Additionally, it employs multi-source reasoning to avoid producing hallucinations, while providing a dialogue that is human-like.

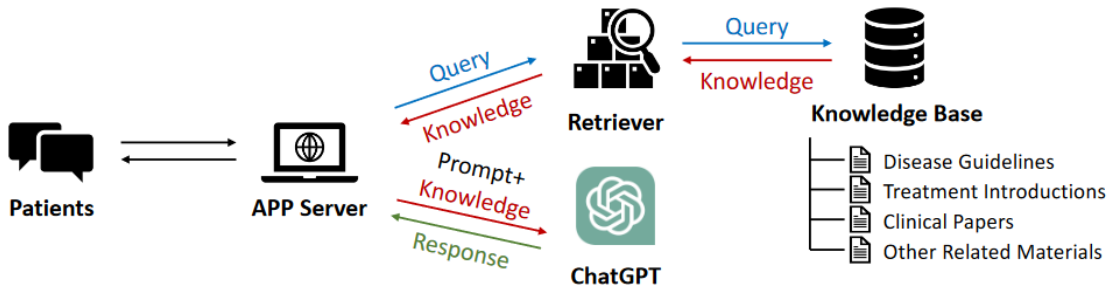


Figure 12: Chat-Orthopedist [78]

[6] https://www.microsoft.com/en-us/bing/apis/bing-web-search-api

BEEDS [94] applies the RALLM approach to literature in biomedicine, using a three-step pipeline (document retriever, document reader, normalizer) to extract event triples for a searched protein or gene and put them in a knowledge database can be saved.

Schumann et al. [74] applies the idea of document retrieval to a text corpus of 709 German regulatory documents to collect relevant regulatory information for an audit. They examine 13 different retrieval variants and 16 search queries, which show that hybrid variants of sparse and dense retrievers perform best.

The RALLM applications presented so far have assumed that they are questions about publicly available knowledge, but there are numerous use cases in which questions about private or proprietary knowledge can be asked. Responding to this requirement, Split Iterative Retrieval (SPIRAL) [3] creates a RALLM with an understanding of private data.

4.2 Applications in Software Engineering

In this section, various applications for RALLMs in the field of software engineering will be presented.

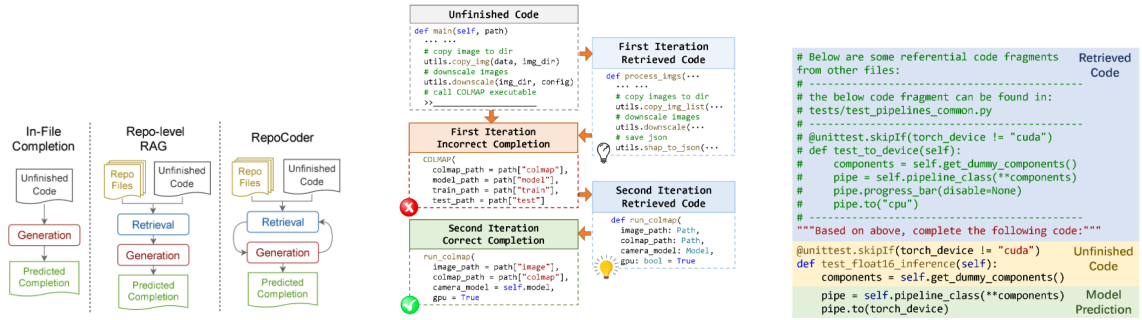
First of all, there are numerous applications for LLMs in software engineering (LLM4SE). Numerous survey papers provide a good overview over models [101], applications [20] and literature [29]. However, the applications presented in this work are limited to the use of LLMs in combination with retrieval systems in software engineering.

The applications of RALLMs in software engineering primarily include classic code generation tasks that are expanded with retrieval systems (e.g. at the level of a repository). But there are also some applications in the areas of requirements engineering, program repairing, code summarization, code search and commit generation. It should be particularly pointed out that the focus is on which applications retrieval methods are used and how they are used, and less on the applications themselves.

4.2.1 Code Generation with Retrieval Methods

Code generation and code completion has the task of generating code from a natural language description or completing unfinished code. RALLM approaches can help with this task. First, different repository-level code generation approaches are presented, with a particular focus on RepoCoder by Zhang et al. [104]. This is followed by code generation approaches that refer to API specifications or code documentation during retrieval.

RepoCoder [104] is a generic framework to complete unfinished code based on the context of the associated repository. This makes it possible to complete code based on dependencies on other files in the repository (including shared utilities, configurations, cross-API invocations) while taking into account naming conventions and coding styles of the respective repository.
RepoCoder uses a retriever to find relevant context for code completion within the repository and an LLM as a generator. Figure 13a shows that, compared to other techniques (especially RAG approach), RepoCoder uses an iterative pipeline of retriever and generator.



(a) Structure (b) Iterations (c) Prompt

Figure 13: RepoCoder [104]

Figure 13b shows a code example for this iteration, in which the COLMAP API is called in the first iteration. However, the predicted parameters are incorrect because the retrieval query was not yet suitable. In a second iteration, the target API signature can be retrieved using the incorrect API call as a new query and the code can be completed correctly.
Based on the code files of the repository $C_{repo} = \{c_1, c_2, ...\}$, a retriever $\mathbb{R}$ searches with the unfinished code X as a query in the repository C_{repo} the most relevant code snippets $C_{ret} = \mathbb{R}(C_{repo}, X)$. Then an LLM $\mathcal{M}$ generates a prediction for the searched code $\hat{Y} = \mathcal{M}(C_{ret}, X)$. Figure 13c shows an example prompt of how the retrieved code snippets C_{ret} are combined with the unfinished code X. Since this is an iterative procedure, the previous prediction $\hat{Y}^{i-1}$ is used as the new retrieval query for the following i-th iteration, so that in the i-th iteration $C^i_{ret} = \mathbb{R}(C_{repo}, X, \hat{Y}^{i-1})$ are returned as relevant code snippets and $\hat{Y}^i = \mathcal{M}(C^i_{ret}, X)$ as a new prediction. $\mathcal{M}$ and $\mathbb{R}$ remain unchanged throughout the entire retrieval process. The retriever $\mathbb{R}$ can be any retriever that, given a query, returns relevant documents (experiments in the paper with sparse and dense retrievers). The generator $\mathcal{M}$ can be any pre-trained LLM (experiments with GPT-3.5-Turbo and CodeGen). The code snippets in the retrieval database are generated using a sliding window approach with a window size of S_w and a sliding size of S_s.
RepoCoder also includes a new benchmark RepoEva from a collection of Python repositories from GitHub, making it the first benchmark with three levels of code completion granularity (line completion, API invocation completion, function body completion). Experiments show that RepoCoder improves existing in-file completion benchmarks by over 10%, especially after at least two iterations. However, they still describe some limitations that leave room for improvements in the future. These include, in particular, limited effectiveness in repositories with low code duplication. Improvements in time efficiency, in choosing the optimal number of iterations or suitable prompt templates are still possible.

In addition to RepoCoder [104], there are other code generation approaches that refer to a repository.

ReACC [53] is a retrieval-augmented code completion framework comparable to RepoCoder, but it is not limited to its own repository in the source code database. It also consists of a retriever and a generation component, but only goes through one iteration

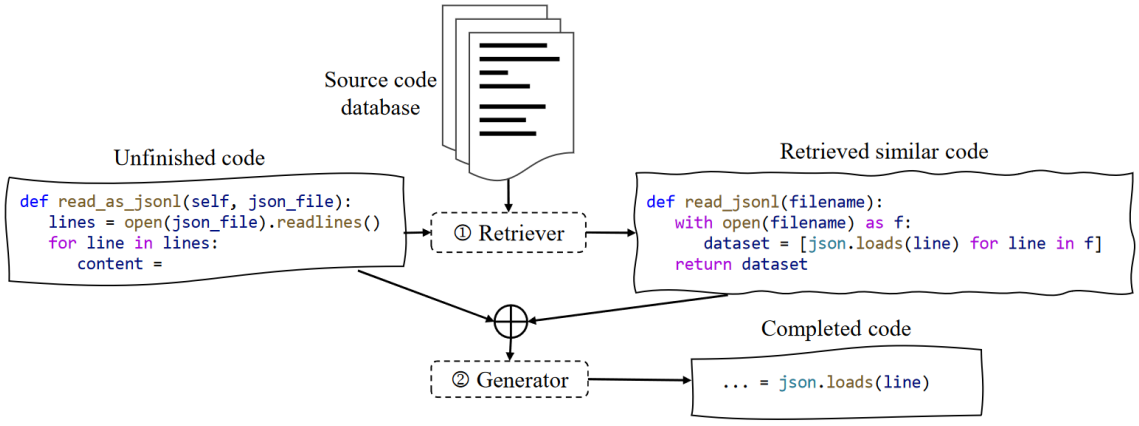


Figure 14: ReAcc [53]

(see middle scheme in Figure 13a). Figure 14 shows an example of how incomplete code can be completed using a retrieved similar code.

RepoFusion (under review) [79] expands the idea of RepoCoder and suggests training code models with the integration of relevant repository context with the Fusion-in-Decoder method in order to benefit from better performance while at the same time being smaller models to benefit.

CodeGen4Libs [49] is an approach to library-oriented code generation. In the first step (import generation stage) it imports import statements from third-party libraries and in the second step (code generation stage) it generates code based on the query and the imports. It can improve previous benchmarks.

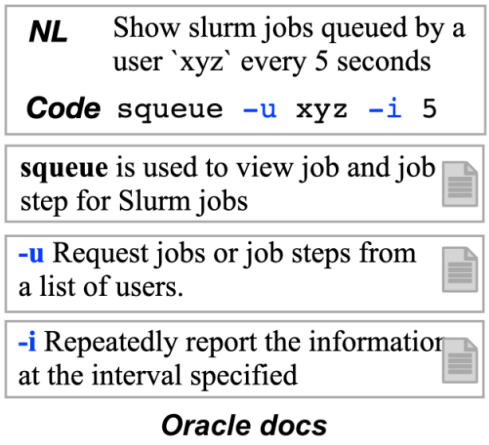


Figure 15: DocPrompting [108]

RepoBench [51] (under review) also represents a new benchmark for repository-level code auto-completion systems with zero-shot learning. It consists of three sub-tasks for the programming languages Java and Python: Retrieval Task (RepoBench-R, ability to retrieve the most relevant code snippets), Code Completion Task (RepoBench-C, predict the next line of code), End-to-End Pipeline Task (RepoBench-P, simulate the complete process of code auto completion).

A Stanford CS224N Custom Project [89] shows that it is in principle possible to use StackOverflow[7] as a code snippet corpus. This expands the amount of accessible knowledge and removes the restriction to specific repositories and data sets for the retrieval step.

[7] https://stackoverflow.com/

After dealing with the repository level-specific code generation approaches, approaches that relate to API specifications or code documentation will now be discussed: APICoder [102] and DocPrompting [108].

DocPrompting [108] is a natural language to code framework (retrieval-then-generate paradigm) that can generate code based on a retrieval of relevant documentation (e.g. code manuals). Given a query in natural language, DocPrompting retrieves relevant parts of the code documentation from an up-to-date documentation pool. It then generates programming code based on the retrieval of the input query (example in figure 15). It is inspired by human programming behavior, where human programmers are inspired by code manuals. In contrast to previous approaches, reference can be made to unseen functions or libraries that were not included in the training data or have changed since then (for example new arguments in the function). It can be applied to any programming language and is not dependent on the underlying LLM. This makes it the first approach to leveraging documentation in models, improving numerous existing benchmarks. They also publish a new benchmark for retrieval-based code generation.

APICoder [102] is an approach comparable to DocPrompting, which enables retrieval (APIRetriever) of the API documentation for private libraries in order to generate code (APICoder) based on it (see Figure 16). Both modules (APIRetriever, APICoder) were trained with data from public libraries and can generalize to private libraries. In addition, three benchmarks (TorchDataEval, MonkeyEval, BeatNumEval) are published for private libraries.

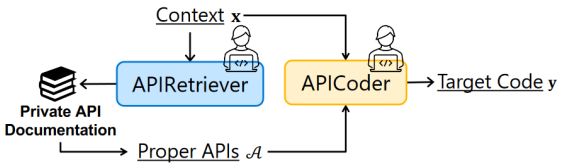


Figure 16: APICoder [102]

CRUSH4SQL [39] is a Text2SQL approach that uses a hallucinated minimal database schema as a query for an information retriever. A two-stage mechanism is used. In a first step, an LLM is used to hallucinate a minimal database schema. This hallucination serves as a bridge between the lexical gap between the tokens of the user query and the actual schema elements. In the second step, the system uses the hallucinated schema to extract a subset of the schema that is closest to the hallucinated schema. Unlike traditional Text2SQL generators, which include and encode the entire schema, this approach is more effective for large databases with thousands of columns, also because it requires fewer computing and storage resources. Since there are currently no benchmark data sets with information retrieval for Text2SQL tasks with schema subsetting, three new benchmark data sets are also being introduced.

4.2.2 Code Search with Retrieval Methods

There are also applications that are limited to code retrieval in the sense of a code search. This makes it more of a retrieval application than a RALLM application.

Deep Code Search (DeepCS) [23] is a Code-Description Embedding Neural Network (CODEnn) based code search tool based on a dense retriever. For the search, code snippets

and natural language descriptions are transferred to a high-dimensional vector space so that the code snippet and the associated description have similar vectors. DeepCS / CODEnn thus takes into account that source code and its natural language description are heterogeneous in their lexical tokens, but are semantically closely linked. Figure 17 shows the architecture of DeepCS, which includes three phases: offline training, offline code embedding, online code search. CODEnn was trained on over 18.2 million Java code snippets.

CodeRetriever [47] learns code semantics at the function level through large-scale code-text contrastive pre-training. In doing so, it reacts to previous weaknesses of token-based approaches to code search. These include the token imbalance in programming languages (keywords or operators appear in many places in the code) and the cross-language representation (challenge to learn a unified semantic representation of the code with the same functionality but using different programming languages). CodeRetriever consists of a text encoder and a code encoder that transforms text and code into separate dense vectors, minimizing the unimodal contrastive loss and the bimodal contrastive loss during training. Experiments show that CodeRetriever performs best compared to all other approaches.

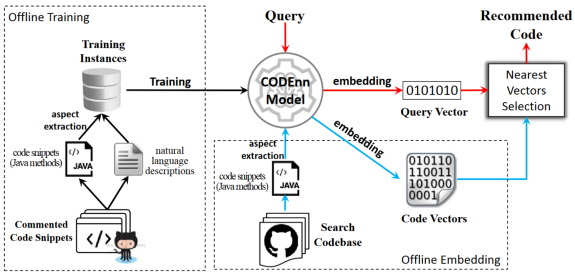


Figure 17: DeepCS [23]

Li et al. [45] also shows how query expansion and query rewriting / augmentation methods explained in Section 3.1 can be applied to code search. This achieves a new state-of-the-art performance.

4.2.3 Code Summarizing with Retrieval Methods

Given a code snippet, code summarization is intended to generate a summary of that code snippet so that software developers can understand code quickly and correctly. Some approaches in which RALLM can help are presented below.

REDCODER [62] is a framework that retrieves relevant code or code summaries from a retrieval database in order to pass them on as a supplement to a model for code generation or code summarization. The consideration at this point should be limited to code summary. To do this, a code snippet is passed as input to a retriever model (SCODE-R, based on the DPR [36] from Section 3.2), which then extracts k relevant summaries from a Database (e.g. GitHub or StackOverflow). The retrieved summary is then concatenated with the original input code and the generator (SCODE-G, a variant of BART) then generates a summary of the code. Experiments show that retrieval methods improve code summarization.

EditSum [46] is a retrieve-and-edit framework for source code summarization comparable to REDCODER. A retriever module (sparse retriever BM25 [69], see Section 3.2) retrieves a similar code snippet from a pre-defined corpus in order to use it as a prototype summary. The prototype serves as a starting point for the edit module (encoder-decoder-architecture), and is then adapted to the semantic information of the input code.

The approaches described so far (REDCODER and EditSum) have the disadvantage that they neither take into account the relationship between the original code and the similar code nor the relationship between the original code and the summary of the similar code, as these are each processed separately and simply concatenated become. READSUM [14] responds to this disadvantage with a Transformer model for source code summarization that uses retrieval augmented techniques. The relevance between the original code and the retrieved code is taken into account using attention-based augmentation and important keywords of the similar summary from the original code are extracted using a fusion network. To learn the relationship between the original code and the retrieval code, a code representation based on a multi-head self-attention mechanism is used, using the Abstract Syntax Tree (AST) sequence at the embedding stage; i.e. instead of being processed separately, sequential and structural processes are processed in a single transformer. Experiments show that READSUM achieves state-of-the-art performance on all evaluation metrics for Java data sets and also outperforms many baselines for Python.

4.2.4 Requirements Traceability with Retrieval Methods

Requirements traceability refers to the links between software artifacts (forward and backward) and helps throughout the entire software evolution process to check the completeness of a software artifact in relation to the requirements, to discover dependencies and thus to ensure the overall quality of a software.

Udagawa [87] discusses early approaches based on vector space information retrieval methods for the automatic restoration of requirements traceability. Similarities between artifacts from the requirements phase and the design phase are measured, whereby the similarity measures depend heavily on the accuracy of the description. Sparse vectors based on inverse document frequency are used.

CERBERUS [18] is also an early approach that attempts to identify the source code in a program in relation to a feature or requirement and uses information retrieval. The requirements or features are viewed as a query and the source code as a document. Sparse vectors are used, which are formed with special consideration of the keywords of the programming language and the inverse document frequency.

YamenTrace [58] is a newer approach to recover and visualize Requirement-to-Code Traceability Links (RtC-TLs), which also uses retrieval methods. To do this, latent semantic indexing (LCI) and singular value decomposition (SVD) are used to search for textual similarity between the code and the requirements on the term-document matrix. A formal concept analysis (FCA) further clusters similar code and requirements.

4.2.5 Commit Message Generation with Retrieval Methods

Commit messages should document and summarize changes in the code and be as meaningful as possible so that a good understanding of the evolutionary history of a software is created. RALLM approaches can also help here.

RACE [76] represents a retrieval-augmented commit message generation method. A similar commit message is initially retrieved as an example. A meaningful commit message is then generated based on the content of the code diff and the similar commit message with the support of an example guide (learns the semantic similarity between retrieved and current code diff). A code diff encoder learns the semantics of code diffs and encodes it in high-dimensional semantic space. Repetitive or redundant commit messages should be avoided. Experiments show that RACE outperforms all baselines. Context-Aware Retrieval-based Deep Commit Message Generation (CoRec) [91] is a comparable approach to RACE, but uses end-to-end training.

COME [27] as a newer approach to commit message generation with retrieval methods uses a fine-grained way to represent code changes in embeddings, thus responding to inappropriate representations of code changes from previous approaches. For this purpose, code change representations are learned in a self-supervised manner in an encoder-decoder neural network. COME achieves state-of-the-art performance on various benchmarks.

4.2.6 Automatic Program Repair with Retrieval Methods

Automatic Program Repair (APR) is designed to reduce manual debugging effort using tools. While conventional APR methods follow the search-based approach and heuristic rules, progress in machine learning has meant that program repair could be learned as a mapping from a buggy source program to a correct target program as a sequence to sequence problem (see for example TFix [4]). These models have limited performance due to their limited number of parameters and the complexity of many bugs, so extending these models with a patch retriever is worthwhile.

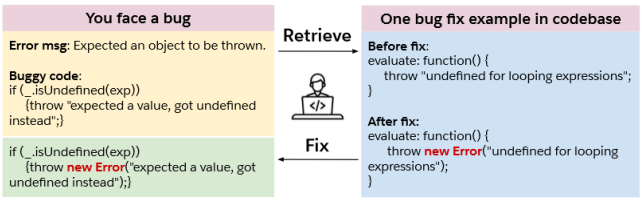


Figure 18: RAP-Gen Bug Example [93]

The Retrieval-Augmented Patch Generation framework (RAP-Gen) [93] is a generic framework for APR with a retrieval system. It uses a hybrid patch retriever, a combination of the sparse retriever BM25 [69] and the dense retriever DPR [36], described in Section 3.2. It consists of an external database that retrieves bug fix pairs to extend the input to the CodeT5 code-specific language model [95] (see Section 2.2). Figure 18 offers an example where the patch retriever is used to search for several similar/relevant bug fixing pairs for a bug, which are then passed on to the CodeT5 patch generator in a ranked list, with the help of which the original bug is fixed. The fix can be verified using unit tests or developer verification. Extensive evaluations show that RAP-Gen outperforms previous state-of-the-art methods on various benchmarks for Java and JavaScript (e.g. 69.3% vs. 78.8% for error removal accuracy on TFix [4]).

5 Discussion and Conclusion

In this seminar paper, current methods and applications for Retrieval-Augmented Large Language Models (RALLMs) were presented. They can solve numerous problems of LLMs and expand them with external knowledge, so that they can also answer questions for which there is no training data and therefore hallucinate less.

It is notable that although research on information retrieval has existed since the 1950s, it has undergone enormous innovation in the early 2020s and the research area has therefore developed in many dimensions (see Section 2). This increase in innovation remains dynamic and is closely associated with the development of LLMs. Nevertheless, there are still numerous approaches for future-oriented scientific research, and there is still room for improvement, particularly when it comes to the factual accuracy of LLMs and RALLMs.

Dealing with private data in RALLMs will remain a challenge. Huang et al. [30] offers a study on privacy risks in retrieval-based LLMs. In order to develop customized in-domain RALLMs, RETA-LLM [48] provides a toolkit for creating a complete pipeline.

Due to the success of RALLMs and due to the large amount of knowledge in the world, it will be relevant to develop RALLMs that can scale to millions of text passages and still produce good results. Pradeep et al. [65] offers some comments and an empirical study. Furthermore, it remains a challenge to reduce the latency of RALLMs and enable multimodal search.

RALLMs usually assume that the knowledge in the external database is factually correct. However, due to numerous fake news on the Internet, this assumption is not always true. Therefore, it remains a challenge that RALLMs can classify external counterfactual knowledge and distinguish correct reliable knowledge from incorrect knowledge. They should also improve their answer quality (less hallucination) and their reference quality (source instead of blackbox). RECALL [52] (under review) is therefore a benchmark for context with counterfactual information in order to be able to build models in the future that can meet this requirement.

All of the challenges described (including scaling, correctness, private data, customization) also exist for RALLM applications in software development. Furthermore, it is noticeable that numerous benchmark data sets for different code generation and summarization tasks have been created in the last 2 years, especially at the repository level (see applications in Section 4.2). As this is a new area of research, these benchmark data sets have yet to become established and widely accepted. Nevertheless, it can be expected that the performance on these benchmark data sets will improve in the future. Despite all the progress on existing benchmark data sets, it remains a challenge to generate high-quality and secure code. For example, a user study [63] shows that the use of AI coding assistance leads to significantly less secure code, which still leaves room for improvement.

References

[1] Anonymous. "Adder: Adapted Dense Retrieval". In: *Submitted to The Twelfth International Conference on Learning Representations*. under review. 2023. URL: https://openreview.net/forum?id=n3kFlvVhJM.

[2] Manoj Ghuhan Arivazhagan et al. "Hybrid Hierarchical Retrieval for Open-Domain Question Answering". In: *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 10680–10689. DOI: 10.18653/v1/2023.findings-acl.679. URL: https://aclanthology.org/2023.findings-acl.679.

[3] Simran Arora et al. "Reasoning over Public and Private Data in Retrieval-Based Systems". In: *Transactions of the Association for Computational Linguistics* 11 (2023), pp. 902–921. DOI: 10.1162/tacl_a_00580. URL: https://aclanthology.org/2023.tacl-1.51.

[4] Berkay Berabi et al. "TFix: Learning to Fix Coding Errors with a Text-to-Text Transformer". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 780–791. URL: https://proceedings.mlr.press/v139/berabi21a.html.

[5] Jonathan Berant et al. "Semantic Parsing on Freebase from Question-Answer Pairs". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Ed. by David Yarowsky et al. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1533–1544. URL: https://aclanthology.org/D13-1160.

[6] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30.1 (1998). Proceedings of the Seventh International World Wide Web Conference, pp. 107–117. ISSN: 0169-7552. DOI: https://doi.org/10.1016/S0169-7552(98)00110-X. URL: https://www.sciencedirect.com/science/article/pii/S016975529800110X.

[7] Tom Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[8] Claudio Carpineto and Giovanni Romano. "A Survey of Automatic Query Expansion in Information Retrieval". In: *ACM Comput. Surv.* 44.1 (Jan. 2012). ISSN: 0360-0300. DOI: 10.1145/2071389.2071390. URL: https://doi.org/10.1145/2071389.2071390.

[9] Danqi Chen et al. "Reading Wikipedia to Answer Open-Domain Questions". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Regina Barzilay and Min-Yen Kan. Vancouver,

Canada: Association for Computational Linguistics, July 2017, pp. 1870–1879. DOI: 10.18653/v1/P17-1171. URL: https://aclanthology.org/P17-1171.

[10] Jiawei Chen et al. *Benchmarking Large Language Models in Retrieval-Augmented Generation*. Accepted to AAAI 2024. 2023. arXiv: 2309.01431 [cs.CL].

[11] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG].

[12] Wenhu Chen et al. "MuRAG: Multimodal Retrieval-Augmented Generator for Open Question Answering over Images and Text". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 5558–5570. DOI: 10.18653/v1/2022.emnlp-main.375. URL: https://aclanthology.org/2022.emnlp-main.375.

[13] Sukmin Cho et al. "Discrete Prompt Optimization via Constrained Generation for Zero-shot Re-ranker". In: *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 960–971. DOI: 10.18653/v1/2023.findings-acl.61. URL: https://aclanthology.org/2023.findings-acl.61.

[14] Yunseok Choi et al. "READSUM: Retrieval-Augmented Adaptive Transformer for Source Code Summarization". In: *IEEE Access* 11 (2023), pp. 51155–51165. DOI: 10.1109/ACCESS.2023.3271992.

[15] Vincent Claveau. "Neural Text Generation for Query Expansion in Information Retrieval". In: *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. WI-IAT '21. Melbourne, VIC, Australia: Association for Computing Machinery, 2022, pp. 202–209. ISBN: 9781450391153. DOI: 10.1145/3486622.3493957. URL: https://doi.org/10.1145/3486622.3493957.

[16] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

[17] Qingxiu Dong et al. *A Survey on In-context Learning*. 2023. arXiv: 2301.00234 [cs.CL].

[18] Marc Eaddy et al. "CERBERUS: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis, and Program Analysis". In: *2008 16th IEEE International Conference on Program Comprehension*. 2008, pp. 53–62. DOI: 10.1109/ICPC.2008.39.

[19] Shahul Es et al. *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. 2023. arXiv: 2309.15217 [cs.CL].

[20] Angela Fan et al. *Large Language Models for Software Engineering: Survey and Open Problems*. 2023. arXiv: 2310.03533 [cs.SE].

[21] 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*. SIGIR '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2288–2292. ISBN: 9781450380379. DOI: 10.1145/3404835.3463098. URL: https://doi.org/10.1145/3404835.3463098.

[22] Luyu Gao et al. "Precise Zero-Shot Dense Retrieval without Relevance Labels". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 1762–1777. DOI: 10.18653/v1/2023.acl-long.99. URL: https://aclanthology.org/2023.acl-long.99.

[23] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. "Deep Code Search". In: *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 2018, pp. 933–944. DOI: 10.1145/3180155.3180167. URL: https://ieeexplore.ieee.org/abstract/document/8453172.

[24] Jiafeng Guo et al. "Semantic Models for the First-Stage Retrieval: A Comprehensive Review". In: *ACM Transactions on Information Systems* 40.4 (Mar. 2022), pp. 1–42. ISSN: 1558-2868. DOI: 10.1145/3486250. URL: http://dx.doi.org/10.1145/3486250.

[25] Daniel Guzman Olivares, Lara Quijano, and Federico Liberatore. "Enhancing Information Retrieval in Fact Extraction and Verification". In: *Proceedings of the Sixth Fact Extraction and VERification Workshop (FEVER)*. Ed. by Mubashara Akhtar et al. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 38–48. DOI: 10.18653/v1/2023.fever-1.4. URL: https://aclanthology.org/2023.fever-1.4.

[26] Donna Harman. "Overview of the First TREC Conference". In: *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '93. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1993, pp. 36–47. ISBN: 0897916050. DOI: 10.1145/160688.160692. URL: https://doi.org/10.1145/160688.160692.

[27] Yichen He et al. "COME: Commit Message Generation with Modification Embedding". In: *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2023. New York, NY, USA: Association for Computing Machinery, 2023, pp. 792–803. DOI: 10.1145/3597926.3598096. URL: https://doi.org/10.1145/3597926.3598096.

[28] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[29] Xinyi Hou et al. *Large Language Models for Software Engineering: A Systematic Literature Review*. 2023. arXiv: 2308.10620 [cs.SE].

[30] Yangsibo Huang et al. “Privacy Implications of Retrieval-Based Language Models”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 14887–14902. URL: https://aclanthology.org/2023.emnlp-main.921.

[31] Gautier Izacard et al. “Unsupervised Dense Information Retrieval with Contrastive Learning”. In: *Transactions on Machine Learning Research* (2022). ISSN: 2835-8856. URL: https://openreview.net/forum?id=jKN1pXi7b0.

[32] Rolf Jagerman et al. *Query Expansion by Prompting Large Language Models*. 2023. arXiv: 2305.03653 [cs.IR].

[33] Zhengbao Jiang et al. “Active Retrieval Augmented Generation”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 7969–7992. URL: https://aclanthology.org/2023.emnlp-main.495.

[34] Jeff Johnson, Matthijs Douze, and Hervé Jégou. “Billion-Scale Similarity Search with GPUs”. In: *IEEE Transactions on Big Data* 7.3 (2021), pp. 535–547. DOI: 10.1109/TBDATA.2019.2921572.

[35] Jean Kaddour et al. *Challenges and Applications of Large Language Models*. 2023. arXiv: 2307.10169 [cs.CL].

[36] Vladimir Karpukhin et al. “Dense Passage Retrieval for Open-Domain Question Answering”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 6769–6781. DOI: 10.18653/v1/2020.emnlp-main.550. URL: https://aclanthology.org/2020.emnlp-main.550.

[37] Omar Khattab et al. *Demonstrate-Search-Predict: Composing retrieval and language models for knowledge-intensive NLP*. 2023. arXiv: 2212.14024 [cs.CL].

[38] Gangwoo Kim et al. “Tree of Clarifications: Answering Ambiguous Questions with Retrieval-Augmented Large Language Models”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 996–1009. URL: https://aclanthology.org/2023.emnlp-main.63.

[39] Mayank Kothyari et al. “CRUSH4SQL: Collective Retrieval Using Schema Hallucination For Text2SQL”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 14054–14066. URL: https://aclanthology.org/2023.emnlp-main.868.

[40] Tom Kwiatkowski et al. “Natural Questions: A Benchmark for Question Answering Research”. In: *Transactions of the Association for Computational Linguistics* 7 (2019). Ed. by Lillian Lee et al., pp. 452–466. DOI: 10.1162/tacl_a_00276. URL: https://aclanthology.org/Q19-1026.

[41] Jakub Lála et al. “PaperQA: Retrieval-Augmented Generative Agent for Scientific Research”. In: *Submitted to The Twelfth International Conference on Learning Representations*. under review, available at https://openreview.net/forum?id=clU5xWyItb and https://arxiv.org/abs/2312.07559. 2023. URL: https://openreview.net/forum?id=clU5xWyItb.

[42] Angeliki Lazaridou et al. *Internet-augmented language models through few-shot prompting for open-domain question answering*. 2023. URL: https://openreview.net/forum?id=hFCUPkSSRE.

[43] Mike Lewis et al. “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, July 2020, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703. URL: https://aclanthology.org/2020.acl-main.703.

[44] Patrick Lewis et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. In: NIPS’20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546. URL: https://dl.acm.org/doi/abs/10.5555/3495724.3496517.

[45] Dong Li et al. *Generation-Augmented Query Expansion For Code Retrieval*. 2022. arXiv: 2212.10692 [cs.SE].

[46] Jia Allen Li et al. “EditSum: A Retrieve-and-Edit Framework for Source Code Summarization”. In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Nov. 2021. DOI: 10.1109/ase51524.2021.9678724. URL: http://dx.doi.org/10.1109/ASE51524.2021.9678724.

[47] Xiaonan Li et al. “CodeRetriever: A Large Scale Contrastive Pre-Training Method for Code Search”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 2898–2910. DOI: 10.18653/v1/2022.emnlp-main.187. URL: https://aclanthology.org/2022.emnlp-main.187.

[48] Jiongnan Liu et al. *RETA-LLM: A Retrieval-Augmented Large Language Model Toolkit*. 2023. arXiv: 2306.05212 [cs.IR].

[49] Mingwei Liu et al. “CodeGen4Libs: A Two-Stage Approach for Library-Oriented Code Generation”. In: *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2023, pp. 434–445. DOI: 10.1109/ASE56229.2023.00159.

[50] Pengfei Liu et al. “Pre-Train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing”. In: *ACM Comput. Surv.* 55.9 (Jan. 2023). ISSN: 0360-0300. DOI: 10.1145/3560815. URL: https://doi.org/10.1145/3560815.

[51] Tianyang Liu, Canwen Xu, and Julian McAuley. “RepoBench: Benchmarking Repository-Level Code Auto-Completion Systems”. In: *Submitted to The Twelfth International Conference on Learning Representations*. under review, available at https://openreview.net/forum?id=pPjZIOuQuF and https://arxiv.org/abs/2306.03091. 2023.

[52] Yi Liu et al. "RECALL: A Benchmark for LLM Robustness against External Counterfactual Knowledge". In: under review, ACL ARR 2023 December Blind Submission, available at https://openreview.net/forum?id=yMHcVZQtP6 and https://arxiv.org/pdf/2311.08147.pdf. 2023.

[53] Shuai Lu et al. "ReACC: A Retrieval-Augmented Code Completion Framework". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 6227–6240. DOI: 10.18653/v1/2022.acl-long.431. URL: https://aclanthology.org/2022.acl-long.431.

[54] Xinbei Ma et al. "Query Rewriting in Retrieval-Augmented Large Language Models". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 5303–5315. DOI: 10.18653/v1/2023.emnlp-main.322. URL: https://aclanthology.org/2023.emnlp-main.322.

[55] Kelong Mao et al. "Large Language Models Know Your Contextual Search Intent: A Prompting Framework for Conversational Search". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1211–1225. DOI: 10.18653/v1/2023.findings-emnlp.86. URL: https://aclanthology.org/2023.findings-emnlp.86.

[56] Vaibhav Mavi, Abulhair Saparov, and Chen Zhao. "Retrieval-Augmented Chain-of-Thought in Semi-structured Domains". In: *Proceedings of the Natural Legal Language Processing Workshop 2023*. Ed. by Daniel Preoțiuc-Pietro et al. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 178–191. DOI: 10.18653/v1/2023.nllp-1.18. URL: https://aclanthology.org/2023.nllp-1.18.

[57] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[58] Ra'Fat Al-Msie'deen. *Requirements Traceability: Recovering and Visualizing Traceability Links Between Requirements and Source Code of Object-oriented Software Systems*. 2023. DOI: http://dx.doi.org/10.12785/ijcds/140123.

[59] Reiichiro Nakano et al. *WebGPT: Browser-assisted question-answering with human feedback*. 2022. arXiv: 2112.09332 [cs.CL].

[60] Rodrigo Nogueira et al. "Document Ranking with a Pretrained Sequence-to-Sequence Model". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Ed. by Trevor Cohn, Yulan He, and Yang Liu. Online: Association for Computational Linguistics, Nov. 2020, pp. 708–718. DOI: 10.18653/v1/2020.findings-emnlp.63. URL: https://aclanthology.org/2020.findings-emnlp.63.

[61] OpenAI et al. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[62] Md Rizwan Parvez et al. “Retrieval Augmented Code Generation and Summarization”. In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Ed. by Marie-Francine Moens et al. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2719–2734. DOI: 10.18653/v1/2021.findings-emnlp.232. URL: https://aclanthology.org/2021.findings-emnlp.232.

[63] Neil Perry et al. “Do Users Write More Insecure Code with AI Assistants?” In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 2785–2799. DOI: 10.1145/3576915.3623157. URL: https://doi.org/10.1145/3576915.3623157.

[64] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. *RankVicuna: Zero-Shot Listwise Document Reranking with Open-Source Large Language Models*. 2023. arXiv: 2309.15088 [cs.IR].

[65] Ronak Pradeep et al. “How Does Generative Retrieval Scale to Millions of Passages?” In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1305–1321. URL: https://aclanthology.org/2023.emnlp-main.83.

[66] Filip Radlinski and Nick Craswell. “A Theoretical Framework for Conversational Search”. In: *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval*. CHIIR ’17. Oslo, Norway: Association for Computing Machinery, 2017, pp. 117–126. ISBN: 9781450346771. DOI: 10.1145/3020165.3020183. URL: https://doi.org/10.1145/3020165.3020183.

[67] Pranav Rajpurkar et al. “SQuAD: 100,000+ Questions for Machine Comprehension of Text”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Ed. by Jian Su, Kevin Duh, and Xavier Carreras. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2383–2392. DOI: 10.18653/v1/D16-1264. URL: https://aclanthology.org/D16-1264.

[68] Ori Ram et al. “In-Context Retrieval-Augmented Language Models”. In: *Transactions of the Association for Computational Linguistics* 11 (2023), pp. 1316–1331. DOI: 10.1162/tacl_a_00605. URL: https://aclanthology.org/2023.tacl-1.75.

[69] Stephen Robertson and Hugo Zaragoza. “The Probabilistic Relevance Framework: BM25 and Beyond”. In: *Foundations and Trends in Information Retrieval* 3 (Jan. 2009), pp. 333–389. DOI: 10.1561/1500000019. URL: https://dl.acm.org/doi/10.1561/1500000019.

[70] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* (1986). DOI: https://doi.org/10.1038/323533a0. URL: https://www.nature.com/articles/323533a0.

[71] Devendra Sachan et al. “End-to-End Training of Neural Retrievers for Open-Domain Question Answering”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Ed. by Chengqing Zong et al. Online: Association for Computational Linguistics, Aug. 2021, pp. 6648–6662. DOI: 10.18653/v1/2021.acl-long.519. URL: https://aclanthology.org/2021.acl-long.519.

[72] Devendra Sachan et al. “Improving Passage Retrieval with Zero-Shot Question Generation”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 3781–3797. DOI: 10.18653/v1/2022.emnlp-main.249. URL: https://aclanthology.org/2022.emnlp-main.249.

[73] Devendra Singh Sachan et al. “Questions Are All You Need to Train a Dense Passage Retriever”. In: *Transactions of the Association for Computational Linguistics* 11 (2023), pp. 600–616. DOI: 10.1162/tacl_a_00564. URL: https://aclanthology.org/2023.tacl-1.35.

[74] Gerrit Schumann, Katharina Meyer, and Jorge Marx Gomez. “Query-Based Retrieval of German Regulatory Documents for Internal Auditing Purposes”. In: *2022 5th International Conference on Data Science and Information Technology (DSIT)*. 2022, pp. 01–10. DOI: 10.1109/DSIT55514.2022.9943943.

[75] Sina Semnani et al. “WikiChat: Stopping the Hallucination of Large Language Model Chatbots by Few-Shot Grounding on Wikipedia”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 2387–2413. DOI: 10.18653/v1/2023.findings-emnlp.157. URL: https://aclanthology.org/2023.findings-emnlp.157.

[76] Ensheng Shi et al. “RACE: Retrieval-augmented Commit Message Generation”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 5520–5530. DOI: 10.18653/v1/2022.emnlp-main.372. URL: https://aclanthology.org/2022.emnlp-main.372.

[77] Weijia Shi et al. *REPLUG: Retrieval-Augmented Black-Box Language Models*. under review, ACL ARR 2023 October Blind Submission, available at https://arxiv.org/abs/2301.12652 and https://openreview.net/forum?id=6z_yPCrdCA4. 2023.

[78] Wenqi Shi et al. “Retrieval-Augmented Large Language Models for Adolescent Idiopathic Scoliosis Patients in Shared Decision-Making”. In: *Proceedings of the 14th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. BCB ’23. Houston, TX, USA: Association for Computing Machinery, 2023. DOI: 10.1145/3584371.3612956. URL: https://doi.org/10.1145/3584371.3612956.

[79] Disha Shrivastava et al. “RepoFusion: Training Code Models to Understand Your Repository”. In: *Submitted to The Twelfth International Conference on Learning Representations*. under review, available at https://openreview.net/forum?id=2drC319yHQ and https://arxiv.org/abs/2306.10998. 2023.

[80] Amit Singhal and I. Google. “Modern Information Retrieval: A Brief Overview”. In: *IEEE Data Engineering Bulletin* 24 (Jan. 2001), pp. 35–43.

[81] Shamane Siriwardhana et al. “Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering”. In: *Transactions of the Association for Computational Linguistics* 11 (2023), pp. 1–17. DOI: 10.1162/tacl_a_00530. URL: https://aclanthology.org/2023.tacl-1.1.

[82] Krishna Srinivasan et al. “QUILL: Query Intent with Large Language Models using Retrieval Augmentation and Multi-stage Distillation”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Ed. by Yunyao Li and Angeliki Lazaridou. Abu Dhabi, UAE: Association for Computational Linguistics, Dec. 2022, pp. 492–501. DOI: 10.18653/v1/2022.emnlp-industry.50. URL: https://aclanthology.org/2022.emnlp-industry.50.

[83] Weiwei Sun et al. “Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 14918–14937. DOI: 10.18653/v1/2023.emnlp-main.923. URL: https://aclanthology.org/2023.emnlp-main.923.

[84] James Thorne et al. “FEVER: a Large-scale Dataset for Fact Extraction and VERification”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 809–819. DOI: 10.18653/v1/N18-1074. URL: https://aclanthology.org/N18-1074.

[85] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[86] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[87] Yoshihisa Udagawa. “An Augmented Vector Space Information Retrieval for Recovering Requirements Traceability”. In: *2011 IEEE 11th International Conference on Data Mining Workshops*. 2011, pp. 771–778. DOI: 10.1109/ICDMW.2011.27.

[88] Ashish Vaswani et al. “Attention is All You Need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964. URL: https://dl.acm.org/doi/10.5555/3295222.3295349.

[89] Shreyas Vinayakumar and Swagata Ashwani. “AI Can Look Up StackOverflow too: Retrieval-Augmented Code Generation”. In: 2023. URL: http://web.stanford.edu/class/cs224n/final-reports/final-report-169502968.pdf.

[90] Ellen M. Voorhees and Dawn M. Tice. "The TREC-8 Question Answering Track". In: *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*. Ed. by M. Gavrilidou et al. Athens, Greece: European Language Resources Association (ELRA), May 2000. URL: http://www.lrec-conf.org/proceedings/lrec2000/pdf/26.pdf.

[91] Haoye Wang et al. "Context-Aware Retrieval-Based Deep Commit Message Generation". In: *ACM Trans. Softw. Eng. Methodol.* 30.4 (July 2021). ISSN: 1049-331X. DOI: 10.1145/3464689. URL: https://doi.org/10.1145/3464689.

[92] Liang Wang, Nan Yang, and Furu Wei. "Query2doc: Query Expansion with Large Language Models". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 9414–9423. URL: https://aclanthology.org/2023.emnlp-main.585.

[93] Weishi Wang et al. "RAP-Gen: Retrieval-Augmented Patch Generation with CodeT5 for Automatic Program Repair". In: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, pp. 146–158. DOI: 10.1145/3611643.3616256. URL: https://doi.org/10.1145/3611643.3616256.

[94] Xing David Wang, Ulf Leser, and Leon Weber. "BEEDS: Large-Scale Biomedical Event Extraction using Distant Supervision and Question Answering". In: *Proceedings of the 21st Workshop on Biomedical Language Processing*. Ed. by Dina Demner-Fushman et al. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 298–309. DOI: 10.18653/v1/2022.bionlp-1.28. URL: https://aclanthology.org/2022.bionlp-1.28.

[95] Yue Wang et al. "CodeT5+: Open Code Large Language Models for Code Understanding and Generation". In: *The 2023 Conference on Empirical Methods in Natural Language Processing*. 2023. URL: https://openreview.net/forum?id=uu60q7MN7g.

[96] Zhiguo Wang et al. "Multi-passage BERT: A Globally Normalized BERT Model for Open-domain Question Answering". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui et al. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 5878–5882. DOI: 10.18653/v1/D19-1599. URL: https://aclanthology.org/D19-1599.

[97] Jason Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 24824–24837. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

[98] Michihiro Yasunaga et al. “Retrieval-Augmented Multimodal Language Modeling”. In: *Proceedings of the 40th International Conference on Machine Learning*. ICML’23. Honolulu, Hawaii, USA: JMLR.org, 2023.

[99] Fanghua Ye et al. “Enhancing Conversational Search: Large Language Model-Aided Informative Query Rewriting”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 5985–6006. URL: https://aclanthology.org/2023.findings-emnlp.398.

[100] Ori Yoran et al. “Answering Questions by Meta-Reasoning over Multiple Chains of Thought”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 5942–5966. URL: https://aclanthology.org/2023.emnlp-main.364.

[101] Daoguang Zan et al. “Large Language Models Meet NL2Code: A Survey”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 7443–7464. DOI: 10.18653/v1/2023.acl-long.411. URL: https://aclanthology.org/2023.acl-long.411.

[102] Daoguang Zan et al. “When Language Model Meets Private Library”. In: *Findings of the Association for Computational Linguistics: EMNLP 2022*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 277–288. DOI: 10.18653/v1/2022.findings-emnlp.21. URL: https://aclanthology.org/2022.findings-emnlp.21.

[103] Yury Zemlyanskiy et al. “Generate-and-Retrieve: Use Your Predictions to Improve Retrieval for Semantic Parsing”. In: *Proceedings of the 29th International Conference on Computational Linguistics*. Ed. by Nicoletta Calzolari et al. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, Oct. 2022, pp. 4946–4951. URL: https://aclanthology.org/2022.coling-1.438.

[104] Fengji Zhang et al. “RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 2471–2484. URL: https://aclanthology.org/2023.emnlp-main.151.

[105] Jianyi Zhang et al. “ReAugKD: Retrieval-Augmented Knowledge Distillation For Pre-trained Language Models”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 1128–1136. DOI: 10.18653/v1/2023.acl-short.97. URL: https://aclanthology.org/2023.acl-short.97.

[106] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2023. arXiv: 2303.18223 [cs.CL].

[107] Wayne Xin Zhao et al. "Dense Text Retrieval Based on Pretrained Language Models: A Survey". In: *ACM Trans. Inf. Syst.* (Dec. 2023). Just Accepted. ISSN: 1046-8188. DOI: 10.1145/3637870. URL: https://doi.org/10.1145/3637870.

[108] Shuyan Zhou et al. "DocPrompting: Generating Code by Retrieving the Docs". In: *The Eleventh International Conference on Learning Representations*. ICLR 2023 notable top 25%. 2023. URL: https://openreview.net/forum?id=ZTCxT2t2Ru.

[109] Yutao Zhu et al. *Large Language Models for Information Retrieval: A Survey*. 2023. arXiv: 2308.07107 [cs.CL].

[110] Honglei Zhuang et al. "RankT5: Fine-Tuning T5 for Text Ranking with Ranking Losses". In: *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 2308–2313. ISBN: 9781450394086. DOI: 10.1145/3539618.3592047. URL: https://doi.org/10.1145/3539618.3592047.