# UniAsk: AI-powered search for banking knowledge bases

Ilaria Bordino
UniCredit, Italy
ilaria.bordino@unicredit.eu

Francesco Di Iorio
UniCredit, Italy
francesco.diiorio@unicredit.eu

Andrea Galliani
UniCredit, Italy
andrea.galliani@unicredit.eu

Alessio Rosatelli
UniCredit, Italy
alessio.rosatelli@unicredit.eu

Lorenzo Severini
UniCredit, Italy
lorenzo.severini@unicredit.eu

## ABSTRACT

Retrieval-Augmented Generation (RAG) has become ubiquitous as a powerful approach to augment Large Language Models with external information, mitigating well-known limitations such as knowledge cut-off and hallucinations. The literature offers a plethora of contributions describing experiments to employ RAG in specific knowledge domains. However, the successful implementation and deployment of end-to-end RAG systems in real-world enterprise settings is way less explored. We present UniAsk, a RAG-based search system designed, developed and fully deployed for the employees of a European bank. We extensively evaluate the performance of our system with real users. Post-launch analysis reports that UniAsk has improved the efficiency of internal banking processes.

## 1 INTRODUCTION

Large Language Models (LLMs) showcase impressive natural language understanding and generation capabilities, but they still face significant limitations that affect their suitability for many domain-specific or knowledge-intensive tasks in natural language processing (NLP). Notable limitations of LLMs include their relying on static and outdated training knowledge, which may produce *hallucinations* [12, 34] in response to queries beyond their training knowledge or requiring up-to-date information, as well as the lack of transparency of their reasoning processes.

Solving these challenges is critical to ensure a widespread adoption of LLM-based applications by real-world enterprises. Hallucinated, outdated, overly-generic answers can clearly compromise the *reliability* of LLM-based chatbots, and therefore their *large-scale use*. For any real-world industry or sector that nowadays wishes to leverage the power of AI to make a better usage of their data, to provide more accurate predictions, to fasten the automatization of their processes or to offer better services to their clients, it is essential to be able to *trust* AI systems with the capability to produce *factually correct* and *up-to-date* information.

The fast-paced nature of many real-world industrial sectors exacerbates the challenges posed by the hallucination and knowledge cut-off limitations of LLMs, increasing the threats to the accuracy and the currency of the advice they may provide. Modern professional sectors are characterized by a rapid evolution, with regulations and trends changing continuously. Thus, AI systems must be capable of providing timely and accurate advice.

*Retrieval-Augmented Generation* (RAG) [17] has emerged as a promising solution to overcome the limitations of LLMs. RAG enhances an LLM by employing a *retrieval* module to gather *relevant information* from external knowledge bases, and using the retrieved information to enrich the prompt with *contextual knowledge* that the LLM can use to *ground* the content to be generated. By referencing authoritative and verifiable sources such as up-to-date regulations or organization insights, RAG effectively reduces the problem of generating factually incorrect content [7, 29]. In addition to mitigating hallucinations, RAG also ensures enhanced timeliness, alignment with domain experts, and improved controllability and explainability of generated responses [35]. Hence, the integration of RAG with LLMs has proven to be a transformative opportunity to foster the adoption of LLMs into real-world applications requiring AI-generated responses that are accurate, up-to-date, and *trustworthy*.

RAG has recently attracted great attention from both researchers and industry practitioners. Researchers have worked on improving RAG from different angles, especially after the pivotal arrival of OpenAI's *ChatGPT*, which led to a flourishing of proposals to enhance the in-context learning capabilities of LLMs.

At the same time, developers from different enterprise sectors have worked on building effective RAG-based tools for domain-specific use cases. Modern *cloud* technology providers such as AWS or Azure have played a crucial role in *enabling* the integration of the RAG paradigm into AI-driven enterprise applications.

The recent literature offers many contributions describing experiments and attempts to employ RAG in specific knowledge domains. Examples include healthcare [13], administration [16], telecommunications [5], driving assistance [11], biomedicine [22], agriculture [2], law [23, 31]. However, we are not aware of works presenting end-to-end RAG systems deployed in a real-world enterprise setting. Furthermore, most of the existing contributions work with English data, for which resources are more abundant.

In this paper we present **UniAsk**, a **RAG-based search system** designed, developed and **fully deployed** for the employees of UniCredit, a pan-European bank headquartered in Italy.

UniAsk aims at providing UniCredit employees with a modern **search** experience, empowered with **generative answering** capabilities, over an internal knowledge base (available in multiple languages) that contains information on a variety of topics, such as *banking applications*, *governance*, *general processes*, and *technical topics* (software tools, mobile devices, online platforms).

UniAsk has been deployed for the **Italian** employees of UniCredit. We have worked with a knowledge base of 59 308 documents in Italian, and released the system to 30$K$ employees.

Prior to the deployment of UniAsk, the UniCredit employees could access the knowledge base through an existing tool that only supported keyword searches, resulting in a very limited experience, which was often unsatisfactory for the users. UniAsk supports a more modern and powerful search experience, as it provides ($i$) better retrieval algorithms, which combine keyword search with vector search and semantic reranking, and ($ii$) a novel generative answering capability that produces answers in natural language.

We have assessed the retrieval and generation performance of UniAsk conducting (*i*) an **automatic evaluation** on *3 500 questions* with ground-truth relevant documents and answers (provided by domain experts), and (*ii*) **two** large **pilot phases** with **real users**. The former pilot involved *200* subject matter experts for *2* months, and collected *3 000* feedbacks. The latter involved *500* employees from retail branches for *1* month, and gathered *11 000* feedbacks.

Post-launch analysis shows that UniAsk allows to **reduce** the number of **tickets** opened to report unsuccessful searches by around 20%. This translates into a faster retrieval of information, and more time for the employees to focus on higher-value activities. We plan to capitalize on the success of UniAsk, and the lessons learned, to adapt our system to other languages and other use cases, so that we can to improve the efficiency of our internal processes and the quality of the services that UniCredit offers to her customers.

To the best of our knowledge, our work offers a *novel* contribution to the literature, presenting an **end-to-end**, **fully-deployed RAG system**, working on **Italian** documents, and providing a practical, demonstrated benefit to a **real-world** use case.

The remainder of this paper is organized as follows. Section 2 introduces our application scenario. Section 3 details the architecture of UniAsk. Section 4 and 5 respectively describe the retrieval and the generation module of UniAsk. Section 6 details our work on guardrails, while Sections 7 and 8 present our experimental evaluation (automatic and with real users). Section 9 describes the steps made to deploy UniAsk to production and Section 10 discusses the related literature. Finally, Section 11 offers our conclusive remarks.

## 2 PROBLEM SCENARIO

UniCredit is a pan-European banking group with more than 75K employees in different countries. Every employee who has an enquiry on Technology, Products and Business, or Legal processes can access to an internal search engine to retrieve information from a large knowledge base containing hundreds of thousands of webpages in multiple languages. In this work we focus on the Italian knowledge base, which contains 59 308 documents.

On average, employees submit thousands of queries per day to the search engine. Whenever an employee is unable to obtain a satisfactory answer for an enquiry of hers, she usually opens a ticket to require the correct information. Every year, thousands of tickets are opened due to search-engine failures, increasing the overall operational cost of handling those tickets.

The existing search engine only performs an exact keyword matching on the documents in the knowledge base. It cannot handle complex questions in natural language. Employees know that an elaborated question will obtain no results, and query the system with one or few keywords. Moreover, the system is uncapable to return answers in natural language. It outputs a ranked list of documents, which the user has to check.

We design UniAsk with a two-fold objective in mind. From one side, we intend to provide the users with a more powerful search experience over internal knowledge bases, allowing them to obtain self-contained natural-language responses for complex questions in natural language, not supported by the existing system. By building a more performant search system, we also aim to reduce the number of tickets opened for unsatisfactory responses, and therefore reduce the overall cost of the related operational process.

## 3 ARCHITECTURE

We now describe the overall architecture of UniAsk, depicted in Figure 1. We deployed UniAsk as a *hybrid architecture* consisting of *microservices* and *serverless* components. Microservices are loosely coupled, fine-grained services, communicating through HTTPS protocol, each performing few tasks. Each microservice is encapsulated within Docker containers that can be deployed differently serving multiple tasks. The *Ingestion* and *Indexing* flow follow a serverless deployment while for the *User Query* flow we choose a Kubernetes[1] deployment workload. To ensure scalability, reliability, deployability and interoperability of our solution, we deployed the services using Microsoft Azure[2].

The **Ingestion** service extracts information from each HTML document in the Knowledge Base (KB). Given that the KB is edited on daily basis, this service is also in charge to keep data updated by polling modifications every 15 minutes. It is deployed on a serverless infrastructure component, triggered by a cron-job mechanism.

The **Indexing** service communicates with the Ingestion service by means of a message queue. Using an event-based trigger, it reads messages posted by the ingester and it feeds the index. The Indexing service parses, chunks and populates metadata for each document of the KB. Notice that we keep a limited chunk size due to the limited LLM prompt length (see Section 5).

Every chunk contains the *title* of the document, the text *content* and *domain*, *section* and *topic* tags provided by the KB editors. We augment the metadata generating via LLM a *summary* of the whole document and a list of *keywords*. The **User query flow** is detailed in Section 4 (Retrieval) and in Section 5 (Generation).

The **FrontEnd** service provides an interface users can interact with. It exposes a search box to query the engine and a feedback form where the user can provide information about the answer quality. The **BackEnd** service is a REST layer exposing endpoints to be called by the frontend. It contains the logic responsible for login and the requests to the Retrieval and Generation services. It stores feedbacks and user actions.

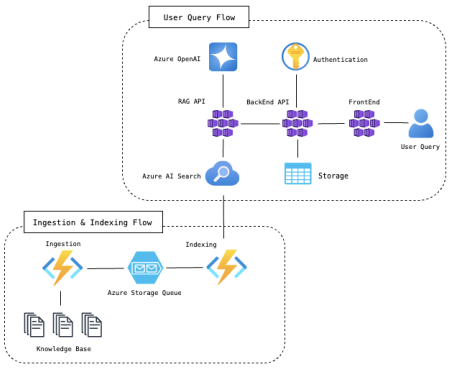


**Figure 1: The architecture of UniAsk**

## 4 RETRIEVAL

The retrieval component of UniAsk is a proper search module that we build and employ to retrieve relevant information from the knowledge base. We build a search index over the document collection, and then we employ retrieval algorithms to retrieve from the index the (fragments of) documents that are most relevant to a user's query.

[1] https://kubernetes.io/
[2] https://azure.microsoft.com/

**Knowledge Base.** The KB contains 59 308 HTML documents in Italian. Documents span different topics such as *banking applications*, *governance*, *general processes* and *technical topics* and tend to be very short: half of them contain just a few sentences. There is a significant amount of content replication, especially among the documents describing procedures or errors, where we may find almost identical content except for specific error or procedure codes. The documents are written by employees for other employees. They are mostly short, containing, on average, 248 words and 7.6 paragraphs. They are characterized by a frequent usage of domain-specific jargon, for which comprehensive vocabularies are not available.

**Index Design and Creation.** We build our index using Azure AI Search[3]. Building the index requires to *split* long documents into *chunks*. We set chunk length to 512 tokens, as the embedding model that we employ for vector search, *text-embedding-ada-002*, is known[4] to perform well on chunks of that size. In our KB, 25% of the documents contain more than 600 tokens. We experimented with two **chunk splitting** strategies. First, we tested Langchain's *RecursiveCharacterTextSplitter*[5], a generic text splitter, which splits a text based on a customizable list of characters, until small-enough chunks are produced. In our experiments, this splitter produced noisy chunks. Therefore, we devised an ad-hoc *HTML-based* strategy, which extracts non-overlapping text chunks from a document by using the start offsets of html paragraphs as splitting points. This ensures that we split a document into coherent fragments, as designed by the human editors who created the page. We *recursively merge* consecutive small chunks until a desired length is obtained.

In Azure AI Search, index fields can be marked with attributes that determine how a field is used, such as whether it's used in full-text search, faceted navigation, sort operations, and so forth. String fields are by default *searchable* and *retrievable*; the latter means that the field can be returned in a search result. We mark document title, chunk content, and summary as retrievable. Domain, topic, section and keywords are marked as *filterable*, to be used for exact matching only. An inverted index is built for each searchable field.

Our index includes separate *vector embeddings* of the document title and the chunk content fields (marked as retrievable). The vector embeddings are obtained by using the *text-embedding-ada-002* from Azure OpenAI.

**Search Algorithm.** To retrieve relevant documents for a query, we employ the **hybrid search**[6] algorithm implemented in Azure AI Search, which combines **full-text** and **vector** search queries.

The *full-text search* module segments an input query into terms using the *it-analyzer-lucene-full* analyzer, which performs stopword removal, sentence splitting, and lower-casing of words. It then retrieves relevant documents for the query by ranking the documents according to the Okapi BM25 [27] ranking function.

The *vector-search* module retrieves documents that are relevant for the query by generating a query embedding and finding the chunks whose title and/or content vectors are closest to the query embedding. For this, we employ the *Hierarchical Navigable Small World (HNSW)* algorithm [21], an Approximate k-Nearest Neighbor algorithm. In our experiments, HNSW and exhaustive k-Nearest Neighbors ($k$-nn) yield similar retrieval performance. We empirically find the best value for $k$, i.e., the number of nearest neighbors returned by HNSW, with dedicated experiments (see Section 7).

The rankings produced by text search (a single ranking) and vector search (one ranking for each vector field) are merged by the **Reciprocal Rank Fusion (RRF)**[7] algorithm, which takes the search results from multiple rankings, assigning to each document/ranking pair a *reciprocal-rank* score that is calculated as $1/(rank + c)$, where *rank* is the position of the document in the list, and $c \geq 1$ is a constant (we use the default value from Azure AI Search, i.e., $c = 60$). The final relevance score assigned to each search result is obtained as the sum of the various reciprocal rank scores *plus* a **semantic reranking**[8] score, obtained with a proprietary multi-lingual, deep-learning model[8] from Bing and Microsoft Research, based on multi-task learning [20], which is integrated into Azure AI Search.

## 5 GENERATION

Once the search module has retrieved the top $m$ relevant document chunks for a given question, the question and the chunks are passed to the generation module, which is responsible for **generating** a proper **answer** in natural language. The answer must respond to the query *based on the relevant documents*, which serve as *context*. UniAsk performs the answer generation task by **querying an LLM**, leveraging its capability to learn new tasks on the fly, with no explicit retraining or parameter updates. In UniAsk, we leverage *gpt3.5-turbo*[9] as the LLM along with its chat completion API.

We use **prompt engineering** to provide the LLM with **task-specific instructions** and with the **context** needed.

We start our prompt by providing the LLM with **general background context**, i.e., by explaining the general scenario and the task the LLM has to deal with. We tell the LLM that it has to serve as a virtual assistant for the UniCredit employees, and that its task is to answer a user's question based on the context (a list of relevant documents retrieved from a domain-specific knowledge base), which is fed to the LLM as input together with the question.

We incorporate the **specific context**, i.e., the top $m$ (with $m = 4$ in the current deployment) document chunks that the search module has retrieved for the user's query, together with **input-format instructions** that help the LLM understand how the input context is formatted. The chosen context format is a JSON list where each document is represented as a dictionary, containing a key identifier, the title and the content of each document chunk.

Next, we provide a sequence of **recommendations** that the LLM must closely follow to provide a **valid answer**. We instruct the LLM that a valid answer must consist of sentences that always cite the relevant chunks from the context that were used as sources, and we give instructions about how to format the citations to reduce variability and increase the likelihood that the LLM uses the context properly. We ask the LLM to always respond in Italian, given that the current UniAsk is intended to be used by Italian employees. We tell the LLM to reply that it does not know the answer to the user's question, when it cannot produce an answer that is clearly based on the provided context. We **repeat** the **instructions** about ensuring that at least one citation is included, and about how to format the citations, **more**

[3] https://learn.microsoft.com/en-us/azure/search/search-what-is-azure-search

[4] https://www.pinecone.io/learn/chunking-strategies/

[5] https://python.langchain.com/v0.2/docs/how_to/recursive_text_splitter/

[6] https://learn.microsoft.com/en-us/azure/search/hybrid-search-overview

[7] https://learn.microsoft.com/en-us/azure/search/hybrid-search-ranking

[8] https://learn.microsoft.com/en-us/azure/search/semantic-search-overview

[9] https://platform.openai.com/docs/models/o1#gpt-3-5-turbo

**than once**. In our experiments, we have observed that repetition of important instructions helps the LLM not to forget the requirements that a valid answer must satisfy.

## 6 GUARDRAILS

We now describe the **guardrails** that we have designed and implemented in UniAsk to ensure that the system is used for its *intended purpose*, and that it *minimizes* risks that may arise from either inappropriate user behavior, or the usage of an LLM.

A primary risk is **hallucination** [12, 34], i.e., the fact that the system may return answers that are not factually correct. For a RAG system, hallucination is a generated answer that is nonsensical or unfaithful to the retrieved context.

Another risk is that the generated response contains *inappropriate language*, possibly causing reputation or legal issues.

Furthermore, we want to avoid any attempt from the users to employ UniAsk *beyond its intended purpose*, to ask questions that are out of scope, or to purse a whatsoever malicious intent.

To mitigate these risks, we enrich UniAsk with **guardrails** [10], a technology that has recently attracted great research effort to enhance LLM-based applications. A guardrail is a shield that we add to instruct our system against specific behaviors or topics.

We implemented **topical guardrails** to prevent UniAsk from generating answers addressing topics that are unrelated to the use case at hand. Our **primary guardrail** works as follows. Once the generation module has returned the answer produced in response to a user's question, the guardrail computes a measure of **similarity** between the generated **answer** and the reference **context** (i.e., the top $m$ chunks returned by the retrieval module). The similarity is computed between the answer and each chunk in the context, returning the maximum score yielded for a chunk as the final score. If the similarity score falls below a predetermined threshold, the guardrail invalidates the answer and the system returns an apology message to inform the user that the system was unable to generate a reliable answer for her question. We adopt a syntactic similarity measure, specifically, ROUGE-L [19]. The threshold on ROUGE-L was heuristically set to 0.15 after conducting exploratory experiments on real user questions.

We also put in place a **secondary guardrail** that invalidates generated answers that contain **no citations to the context**. This idea was suggested by preliminary experiments, in which we noticed that whenever the generated answer did not contain at least one valid citation to the context, the answer was indeed hallucinated.

For both guardrails, we further add a special handling of the generated answers that end with a request for further details, because UniAsk is intended to return a self-contained answer to any input question. When this happens, we raise a **clarification requirement** guardrail, which invalidates the answer and invites the user to reformulate her question with more details.

We also run the *Azure Content Filter* [10] to detect and block harmful content, such as inappropriate language, in the question.

Whenever a guardrail invalidates a generated answer, the system still displays the whole document list for the user to check. We consider the triggering of a guardrail as a failure of the generation module, not of the whole system. The document list might still be relevant, thus we leave the user the chance to check it out.

[10] https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/content-filter

We tested the guardrails on real questions, assessing multiple runs to account for the non-determinism of the LLM, and asking human experts to review blocked answers (see Section 8).

## 7 AUTOMATIC EXPERIMENTAL EVALUATION

Deploying a system in a real-world industrial setting requires a careful pre-go-live evaluation. We tested UniAsk conducting ($i$) an automatic evaluation on real datasets of questions posed by employees, and ($ii$) a live evaluation with real users. This section presents the former, while the latter is reported in Section 8.

For automatic evaluation we built two **datasets** mimicking different querying behaviors: the **human** dataset and the **keyword** dataset. Due to legal constraints, the datasets cannot be made publicly available.

The **human dataset** contains 2700 *real-world* questions, compiled by human experts from the team responsible for the pre-existing search engine. The domain experts were asked to provide *real* and *relevant* questions pertinent to the different topics in the knowledge base, leveraging their understanding of the matters that are known to attract a strong interest and are frequently asked by the employees. For each authored question, the expert was asked to provide a *ground-truth answer* in natural language, and one or more *links* to the documents in the knowledge base that contain the information required to answer the question.

The **keyword** dataset was collected to evaluate how UniAsk performs when asked the typical questions that are presented to the pre-existing system. It contains 800 *keyword-style* questions that were randomly sampled among the frequent queries in the log of the previous system. We used a log spanning one year of data between 2023 and 2024. For each keyword query, we asked our domain experts to provide one or more links to *ground-truth* documents containing the required information. A ground-truth answer in natural language was not collected for these queries, since they just aim at finding a document in the knowledge base.

We split both datasets in two parts: *validation* (2/3 of queries) and *test* (1/3 of queries). We used the validation datasets to tune our approach during the development phase, which was carried out in *agile* mode, and the test datasets for pre-deployment evaluation. We report below our assessment of the performance of UniAsk with respect to ($i$) **retrieval** of documents relevant to a user's question. and ($ii$) **generation** of a natural-language answer for the question.

**Retrieval.** As we stated in Section 4, our retrieval module uses Azure AI's Hybrid Search with Semantic Reranking (HSS) as the **algorithm** of choice to retrieve documents that are relevant for a user's query. The text search component retrieves $n = 50$ documents that are most similar to the input query. For vector search, we set $K = 15$ as the number of most similar documents returned by the Approximate Nearest Neighbor (ANN) algorithm. The value of $K$ was set after exploring several choices ($K \in \{3, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$) on both our validation datasets. HSS merges the 50 documents returned by text search with the 15 returned by vector search, employing Reciprocal Rank Fusion (RRF), and producing a final ranking of 50 documents.

We evaluate the retrieval performance of UniAsk by using standard information retrieval **metrics** such as: precision@n, recall@n, binary hit rate @n, and MRR. For a given question, the binary hit rate@n is = 1 if the top $n$ results returned by the retrieval module contain at least 1 relevant result, 0 otherwise. The average binary hit rate@n over a query dataset represents

| Metric | Human Test Dataset | | | Keyword Test Dataset | | |
|---|---|---|---|---|---|---|
| | Prev. | UniAsk | % Var | Prev. | UniAsk | % Var |
| p@1 | 0.3557 | 0.3840 | 8.0 ↑ | 0.5490 | 0.5099 | −7.1 ↓ |
| p@4 | 0.1195 | 0.1979 | 65.6 ↑ | 0.3006 | 0.3291 | 9.5 ↑ |
| p@50 | 0.0104 | 0.0312 | 200.0 ↑ | 0.0350 | 0.0683 | 95.1 ↑ |
| r@1 | 0.0665 | 0.3753 | 464.4 ↑ | 0.3393 | 0.3316 | −2.3 ↓ |
| r@4 | 0.0886 | 0.5764 | 550.6 ↑ | 0.6332 | 0.5432 | −14.2 ↓ |
| r@50 | 0.0969 | 0.7899 | 715.2 ↑ | 0.8275 | 0.7885 | −4.7 ↓ |
| hit@1 | 0.3556 | 0.3840 | 8.0 ↑ | 0.5490 | 0.5099 | −7.1 ↓ |
| hit@4 | 0.4693 | 0.5889 | 25.5 ↑ | 0.7895 | 0.7352 | −6.9 ↓ |
| hit@50 | 0.5102 | 0.7967 | 56.2 ↑ | 0.8977 | 0.8792 | −2.1 ↓ |
| MRR | 0.0795 | 0.4892 | 515.3 ↑ | 0.6504 | 0.6236 | −4.1 ↓ |

**Table 1: Retrieval performance of UniAsk and of the pre-existing search system (dubbed Prev.), on the test datasets.**

the fraction of queries for which at least one relevant document is retrieved, providing an overview of the system's ability to return relevant results across all queries. It is easy to understand for business experts.

Table 1 reports the retrieval performance on UniAsk on the **human** test dataset and the **keyword** test dataset, compared to the pre-existing search engine, which serves as our internal baseline. The reported results are averages on the questions for which a non-empty document list was obtained. UniAsk was able to retrieve non-empty results for *all* queries in both datasets. The baseline was capable to serve 98.6% of the keyword questions, as expected, since those queries were extracted from its own log. However, the previous system performed very poorly on the human questions: it retrieved non-empty results for just 19.1% of those questions, failing to return anything for the remaining 81.9%. This is a first testament of the superiority of UniAsk. Our system discloses the possibility to formulate complex natural-language questions, which the pre-existing system is unable to serve in most of cases.

| Metric | Human Test Dataset | | Keyword Test Dataset | |
|---|---|---|---|---|
| % var wrt HSS | Text Search | Vector Search | Text Search | Vector Search |
| p@1 | −50.8 | −39.1 | −11.7 | −21.3 |
| r@1 | −51.2 | −39.5 | −10.3 | −18.6 |
| r@4 | −41.6 | −30.1 | −10.5 | −17.4 |
| r@50 | −43.6 | −9.9 | −4.7 | −3.2 |
| hit@1 | −50.8 | −39.1 | −11.7 | −21.3 |
| hit@4 | −41.3 | −30.0 | −10.8 | −15.8 |
| hit@50 | −42.8 | −9.6 | −2.7 | −2.5 |
| MRR | −44.5 | −30.6 | 5.5 | −0.6 |

**Table 2: Ablation study on the components of Hybrid Search**

Further evidence of the superiority of UniAsk can be found in the results reported in Table 1. For each metric/dataset pair, the table reports the percentage improvement of UniAsk over the baseline system. The two search systems perform almost comparably on the keyword dataset, with UniAsk actually achieving slightly worse results with respect to several of the reported metrics. However, in most of cases the percentage loss is below 10%, and our business experts considered this result acceptable in light of the way superior performance that the new system yields on natural-language questions. Not only the baseline could only retrieve results for 19.1% of the human queries: even when the baseline did retrieve results, UniAsk outperformed it with respect to all metrics, and with improvements that are way more significant in magnitude than those obtained on the keyword dataset. In particular, the improvement in recall and MRR is impressively in the order of +500%.

We have validated the choice of HSS by checking how the two components of Hybrid Search, i.e., Text Search and Vector Search perform separately. Table 2 reports the results of this ablation study. For each dataset/algorithm combination, the table reports the percentage variation with respect to HSS for each retrieval metric. Both algorithms perform worse than HSS when employed alone. On the human dataset, the loss is more significant when Text search is used: the questions drafted by human editors have a more complex structure, and require the contribution of Vector Search to better handle semantic similarities. Conversely, Text Search yields lower loss on all metrics for the keyword queries, where syntactic matching contributes more to the final ranking.

In the attempt to further improve the performance of our retrieval module, we have experimented with several ideas. We have tested 3 **query-expansion** variants: (*i*) *QGA*, which asks the LLM to generate an answer for the input query, with no relevant context, and then performs the retrieval step on the query expanded with the generated answer; (*ii*) *MQ*1, which asks the LLM to generate multiple queries related to the input query, and then performs a multi-query hybrid search; (*iii*) *MQ*2, which also asks the LLM to generate multiple related queries for the initial question, but then performs a standard hybrid search on the text concatenation and the average embedding of all queries. We have attempted to **boost term matches** on the **titles** of documents with a multiplicative weight ($T = \{5, 50, 500\}$). We also tried to **enrich** the **index** with **keywords** extracted by the LLM from the title of documents (*HSS-KT*), or from title *and* content (*HSS-KTC*).

| % var wrt HSS | Query Expansion | | | Boosting match on title | | |
|---|---|---|---|---|---|---|
| | QGA | MQ1 | MQ2 | T5 | T50 | T500 |
| p@1 | −15.3 | −4.9 | −3.4 | 2.0 | 0.4 | 0.3 |
| r@1 | −15.4 | −1.4 | 3.6 | 1.7 | 0.0 | 0.0 |
| r@4 | −15.0 | −4.0 | −3.4 | 0.7 | −1.8 | −1.5 |
| r@50 | −15.5 | −9.2 | −5.8 | −1.0 | −5.0 | −5.0 |
| hit@1 | −15.3 | −4.9 | −3.4 | 2.0 | 0.4 | 0.3 |
| hit@4 | −15.0 | −3.9 | −3.4 | 0.8 | −0.9 | −1.2 |
| hit@50 | −14.9 | −8.8 | −5.3 | −1.1 | −4.3 | −4.8 |
| MRR | −14.5 | −5.0 | −3.2 | 0.9 | −0.9 | −1.2 |

**Table 3: (A) query expansion (B) Tuning scoring profiles to boot a text match in title. (Dataset: Human Test Dataset)**

Tables 3 and 4 summarize the results of these experiments, reporting the percentage variation that the mentioned variants yield on the retrieval metrics with respect to the HSS algorithm. Surprisingly, none of these variants yielded significant improvements, thus we did not include them in the final retrieval module.

**Generation and guardrails.** Evaluating the quality of generated answers is crucial in a RAG framework. We thoroughly investigate which metrics can be used to assess how well an answer generated by the LLM replies to a given question. We assume that for an input question, a ground-truth answer is available, in addition to the top $m = 4$ document fragments that are retrieved by the search module as the most relevant to the question. One of the most used metrics in the literature is **groundedness** [18], which evaluates whether an answer is stating facts that are present in a given context. It also leverages an LLM, which is fed a question and the contexts retrieved from the search module, and returns a score representing the extent to which the answer is coherent with the contexts. In our automatic evaluation, groundedness

failed to return meaningful results in the large majority of cases. For this reason, we deferred the assessment of generation perfomance to the tests with real users (Section 8). In Table 5 we report the accuracy of guardrails on the **human** test dataset, i.e., the percentage of questions improperly blocked by the guardrails and the Azure Content Filter. Less than the 6% of the questions were incorrectly blocked by the guardrails.

| % var | Human Test Dataset | | Keyword Test Dataset | |
|---|---|---|---|---|
| wrt HSS | HSS-KT | HSS-KTC | HSS-KT | HSS-KTC |
| p@1 | 0.2 | −1.6 | 0.5 | 0.0 |
| p@4 | −0.6 | 0.5 | 0.0 | 2.1 |
| p@50 | 0.0 | −0.8 | 1.4 | 0.3 |
| r@1 | 0.2 | −1.7 | 0.4 | −1.2 |
| r@4 | −0.7 | 0.3 | −0.3 | 1.4 |
| r@50 | −1.0 | −0.7 | −1.0 | −0.2 |
| hit@1 | 0.2 | −1.6 | 0.5 | 0.0 |
| hit@4 | −0.7 | 0.3 | 0.4 | 1.7 |
| hit@50 | −1.0 | −0.7 | −0.3 | 0.0 |
| MRR | −0.2 | −0.9 | 0.2 | 0.3 |

**Table 4: Enriching the index with keywords**

| Guardrail Type | # Answers |
|---|---|
| *Generated answers (no guardrails)* | 94.8% |
| *Citation guardrail* | 3.5% |
| *Rouge guardrail* | 1.1% |
| *Require clarification guardrail* | 0.2% |
| *Azure OpenAI Content Filter* | 0.5% |

**Table 5: Answer generation rate on the Human Test Dataset**

## 8 EVALUATION WITH REAL USERS

Testing with *real users*, i.e., actual employees, played a crucial role for the success of UniAsk. Before the official deployment, we carried out three test phases: Pilot Phase 1, Pilot Phase 2, and UAT Phase. Each phase had different timings, users and objectives.

**Granular Feedback.**The goal of our tests was to improve the overall performance of our system, in terms of accuracy of the generated answers, as well as user satisfaction. With this aim in mind, we added a pop-up modal to the frontend of UniAsk, to gather granular feedbacks from users.

We asked the following questions:

(1) *Was the answer helpful?*
(2) *Did the system retrieve relevant documents for your question?*
(3) *Rating experience (1 to 5)* : We marked 1 and 2 as negative scores and 3, 4 and 5 as positive scores;
(4) *Links to relevant documents*: We gave our users the chance to provide links to the document(s) containing the answer to the question;
(5) *Additional comments*: We gave our users the possibility to provide further comments about the quality of the search experience.

We found the last two fields extremely useful to gather ground-truth documents and answers for questions on which the system had failed, as well as to collect helpful insights that could leveraged to better understand the reasons of such failures.

**Corner Cases**. In addition to granular feedbacks, we asked our *subject matter experts* (SMEs) to compile a catalogue of *500 corner-case questions* for which a wrong answer would be deemed *unacceptable*, as it could trigger an operative or reputation risk for the bank. We thoroughly analyzed this dataset to get insights about error patterns in the generated answers. The continuous iteration with SMEs resulted in several releases of newly fine-tuned versions of UniAsk. For each release, new tests were performed on the corner cases, to check that all known edge cases were properly tackled.

We now provide details about the three testing phases.

**Phase 1: Pilot with Subject Matter Experts.** Our first test phase lasted *2 months* and involved *200 SMEs*, who were asked to extensively test UniAsk with questions regarding topics in their area of expertise. The intuition for starting with SMEs, was, as imaginable, that they would be more capable to leverage domain knowledge to provide constructive feedback for unsatisfactory answers, as well as useful insights to improve the system. During this phase, we received and reviewed SMEs' feedback on a weekly basis. In the very first round of reviewing, we realized that the domain experts were still mostly querying the system with keyword-style questions, instead of asking questions in natural language. This was due to their longstanding practice with the previous search engine, which they knew to fully function only with keyword requests. We promptly reacted to this issue by preparing guidelines about how to properly use UniAsk. We used those to train the SMEs.

During Phase 1, two different releases of UniAsk were deployed, mostly improving the search index. For the first release, we collected *3000 feedbacks* for *6000* prompted *questions*. For 75% of such questions, UniAsk generated a proper answer with citations to documents in the KB, while 25% triggered guardrails. 77% of the proper answers received a positive evaluation. The number of triggered guardrails was higher than we expected based on automatic evaluation. This was due to a bug that we fixed for the second release. The second release delivered a *proper answer* with citations for *90%* of the questions, of which *78%* got a positive evaluation.

**Phase 2: Pilot with Branch Users.** The second pilot lasted *1 month*, and involved *500 branch users*, i.e., employees working in the retail branches of the bank, who offer face-to-face and automated services to customers. We selected branch employees with different job roles, and geographically spread across all Italy. Branch users for this pilot were picked among the most active in using internal tools, assuming they would be more willing to leave feedbacks. Capitalizing on the experience with SMEs, we trained in advance our branch users to test the system with questions in natural language.

During phase 2, we deployed two further releases, improving the search index and the guardrails. We interacted with branch users on a daily basis, collecting more than *11000 feedbacks*. The percentage of answers with proper citations to documents in the KB kept the same pace (*91% overall*), while positive feedbacks touched a peak *84%*. We analyzed a sample of the negative feedbacks, finding that unsatisfactory answers could mostly be mapped onto two cases: (*i*) the question was too generic and the LLM was unable to provide an answer with citations to documents in the KB (causing the citation guardrail to rise), or (*ii*) the cited documents had strong overlap with other documents, which caused confusion and led the LLM to overlook aspects that were important for the answer.

**Phase 3: User Acceptance Test (UAT)** In the UAT phase, we compiled a **uat dataset** of *210 questions* to be tested for the very last *pre-deployment* approval. The dataset was composed as follows:

- *70 questions in natural language*, sampled from the **human** dataset among those that were more similar (as per Jaccard similarity of non-stop terms) to frequent queries in the log of the previous system, to ensure we were including relevant questions;
- *50 questions in natural language*, chosen by SMEs: 30 were formulated from scratch, and 20 were picked from feedback logs (4 questions for each possible value score in the range $[1-5]$);
- *50 keywords queries*: selected from the **keyword** validation dataset, as the most frequent in the 2023 log of the previous system;
- *10 queries not relevant* to the topics in the KB, selected from the catalogue of corner cases, to test the triggering of guardrails;
- *20 error-code queries*, randomly picked from the SMEs' list;
- *10 special cases* (lower/upper case, missing words, and duplicates).

The results of UAT were manually reviewed by SMEs. UniAsk achieved 87% of correct answers, 89% of guardrails triggered successfully, and a 3% of guardrails improperly triggered.

The main take-away of our tests with real users is that to promote a widespread adoption of UniAsk, we need to *educate the employees* to a radical change in the way they query search systems. In our bank, employees have used a keyword-based search engine for 20 years. Now, we need to train them so that they can appreciate the possibilities disclosed by a system that can be queried in natural language. We are building tutorials to help our users understand how to effectively use the system.

## 9 DEPLOYMENT TO PRODUCTION

This section illustrates the path that we have followed to successfully deploy UniAsk to production, detailing our choices and our work on aspects such as infrastructural resources and application environments, security, devops, scalability and monitoring.

**Infrastructure, Resources & Environments**. UniAsk is fully deployed in a cloud infrastructure, interacting with our internal on-premise components. The application components can be grouped into four distinct environments: *Workbench*, *DEV* (Development), *QA* (Quality), and *PROD* (Production). The Workbench provides all the needed tools for preliminary experiments and data exploration. It includes Storage, Hybrid Search Database, LLM Hosting Service, Ops Services for experiment tracking, and metrics and notebooks for seamless data exploration. The other three environments act as standard promotion environments for application deployment. They are equipped with all the resources listed in the UniAsk architecture, including a dedicated namespace in an AI-specific Kubernetes cluster with dedicated nodes, Storage, Hybrid Search Database and LLM Hosting Service. The application environments differ in the tiering and sizing of resources: DEV is equipped with minimal resources, whereas QA and PROD are exactly equivalent.

**Security**. The cloud infrastructure is secured according to internal cloud standards, which include private network integration and no internet access for the resources. We apply *in-transit* and *at-rest* encryption, which is managed with an internal key management system. The development team is granted access to the cloud infrastructure with a *least-privilege* approach. A dedicated *role-based access-control* system segregates accesses and roles. At application level, prior to production deployment, we perform vulnerability assessments on the code base and penetration tests on the exposed APIs to highlight vulnerabilities.

**DevOps**. Application development and deployment adhere to the internal DevOps best practices. Continuous-Integration (CI) components are triggered via an on-prem DevOps tool at every commit, providing standard interfaces for package build, test, security checks and container build. For Continuous-Deployment (CD) components a dedicated code base is implemented, leveraging on a mixture of Infrastructure-as-code and Kubernetes deployment tools, depending on the target deployment resource and infrastructure.

**Scalability**. To assess scalability, we perform load tests picking the LLM service as the rate limiter for the overall application, given that the LLM inference is the computationally heaviest and most expensive step of our application flow. We treat UniAsk as an open system, where there is no control over the number of concurrent users. Users keep arriving, regardless of the number of concurrent users already in the system. If the system starts slowing down at any point, users who are already in the system will take more time to complete their journey. Figure 2 shows


**Figure 2: Report for a load test on the LLM service of UniAsk.**

the results of a load test on the LLM service of UniAsk. The test consists in continuously hitting the LLM resource with requests during a 60-minute interval, with an initial user amount rate of 1 per second and a target user amount rate of 3 per second. Each request has 7200 tokens in total. The test yields 267 failed queries out of a total of 7200 requests. We use simple calculations based on the load test results to empirically set the token rate limit for the LLM resource.

**Monitoring**. In order to allow a proper monitoring of the health, performance, and usage of UniAsk, we have created a dashboard that directly queries the logs of the various microservices, and collects all the statistics that we need for monitoring purposes.

Figure 3 shows a page of our dashboard, reporting the number of users, the number of feedbacks provided, the average response time, and the number of failed requests and triggered guardrails.

## 10 RELATED WORK

This paper introduces a real-world, end-to-end, fully-deployed RAG system, working on Italian documents. To the best of our knowledge, the recent literature does not yet include any contribution describing a system like ours. Our literature review surveys RAG frameworks for specific domains, LLMs built for

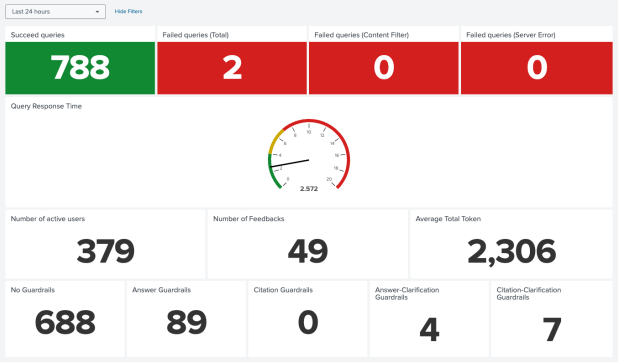


**Figure 3: An example page of the monitoring dashboard.**

the Italian language, as well as recent discussions about limitations and/or possible improvements for RAG systems.

**RAG frameworks and case studies for specific domains**. The recent literature offers a plethora of contributions attempting to adapt RAG to specific knowledge domains. Kawashima et al. [16] introduce a RAG system for administrative documents. Jiang et al. [13] describe a case study for the medical domain. Bornea et al. [5] introduce *Telco-RAG*, an open-source RAG framework designed for the telecommunications sector. Hernandez-Salinas et al. [11] propose IDAS, a RAG-based driving assistant that helps reading a car manual. Matsumoto et al. [22] have developed Kragen, an open-source RAG framework enhanced with knowledge graphs for question answering in the biomedical domain. Balaguer et al. [2] present a pipeline for RAG and fine-tuning of LLMs, with a case study on an agricultural dataset. Wiratunga et al. [31] and Moreira et al. [23] have designed rag-based methods for the legal domain. Xu et al. [32] design a knowledge-graph enhanced RAG based on historical tickets. However, to the best of our knowledge, we are not aware of works presenting end-to-end RAG systems that have been successfully deployed in a real-world enterprise setting.

**LLMs and Generative Applications for the Italian Language**. The last few years have witnessed increased research and development efforts around the Italian language, producing Italian LLMs such as LLamantino [4], Fauno [1], Camoscio [28], and LLaMAntino-3-ANITA-8B-Inst-DPO-ITA [25]. RAG systems are being increasingly built to deal with Italian data. For example, Siragusa et al. [30] have created Unipa-GPT, a chatbot that assists students in choosing a bachelor/master degree course at the University of Palermo. Nevertheless, the recent literature does not yet include end-to-end enterprise RAG applications fully working on Italian, like ours.

**Limitations and possible improvements of RAG systems.** Nguyen et al. [24] investigate the impact of *domain-specific* model *fine-tuning* and *reasoning* mechanisms on the performance of Q&A and RAG systems powered by LLMs. They find that combining a fine-tuned embedding model with a fine-tuned LLM achieves better accuracy than generic models. Baldazzi et al. [3] propose an approach to *fine tune* LLMs for financial markets through *ontological reasoning* on enterprise knowledge graphs. Cuconasu et al. [8] conduct a systematic analysis of the *retrieval* strategy of RAG systems, reporting that the highest-scoring retrieved documents that are not directly relevant to the query negatively impact the effectiveness of the LLM, while adding random documents in the prompt improves accuracy. Raina and Gales [26] propose a zero-shot adaptation of standard dense retrieval steps for more accurate chunk recall. Dong et al. [9] introduce G-RAG, a graph-based context-informed *reranker* for RAG, which combines connections between documents and semantic information. Zeng et al. [33] present an empirical evaluation of the *robustness* of RAG systems to novel *privacy* attack methods. Bruckhaus [6] discusses the challenges of implementing RAG in enterprises, and proposes an evaluation framework for enterprise RAG. Finally, recent works [14, 15] investigate how to properly leverage the increasing capacity of LLMs to process longer input, proposing approaches to optimize Long-RAG frameworks.

## 11 CONCLUSION AND FUTURE WORK

We have presented UniAsk, a RAG-based search system designed, developed and deployed for the employees of UniCredit, a pan-European bank. To the best of our knowledge, this is the first real-world, end-to-end, fully-deployed enterprise RAG system, working on Italian documents, being presented to the literature. We have described real-world deployment challenges and reported extensive experiments with real questions and real users, which demonstrate the performance of our system. Post-launch analysis confirms that UniAsk improves the efficiency of internal operational processes.

We plan to enhance our system under many different directions. We will test further improvements for the *retrieval* module, e.g., fine tuning the embedding model with internal data, or by using embedding adapters. For the *generation* module, we will assess the benefit of using longer context and the impact that multi-modal LLMs may bring to better interpret the information enclosed in documents, including embedded images. We will strengthen our guardrails with more sophisticated approaches for hallucination detection and mitigation. We will consider building a knowledge graph to support guiding the generation via ontological reasoning.

## ACKNOWLEDGMENTS

We are very thankful to Microsoft ISE team for their invaluable support during the early design and implementation stages of our system.

## REFERENCES

[1] Andrea Bacciu, Giovanni Trappolini, Andrea Santilli, Emanuele Rodolà, and Fabrizio Silvestri. 2023. Fauno: The Italian Large Language Model that will leave you senza parole! arXiv:cs.CL/2306.14457

[2] Angels Balaguer, Vinamra Benara, Renato Luiz de Freitas Cunha, Roberto de M. Estevão Filho, Todd Hendry, Daniel Holstein, Jennifer Marsman, Nick Mecklenburg, Sara Malvar, Leonardo O. Nunes, Rafael Padilha, Morris Sharp, Bruno Silva, Swati Sharma, Vijay Aski, and Ranveer Chandra. 2024. RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture. arXiv:cs.CL/2401.08406

[3] Teodoro Baldazzi, Luigi Bellomarini, Stefano Ceri, Andrea Colombo, Andrea Gentili, and Emanuel Sallinger. 2023. Fine-tuning Large Enterprise Language Models via Ontological Reasoning. arXiv:cs.CL/2306.10723

[4] Pierpaolo Basile, Elio Musacchio, Marco Polignano, Lucia Siciliani, Giuseppe Fiameni, and Giovanni Semeraro. 2023. LLaMAntino: LLaMA 2 Models for Effective Text Generation in Italian Language. arXiv:cs.CL/2312.09993

[5] Andrei-Laurentiu Bornea, Fadhel Ayed, Antonio De Domenico, Nicola Piovesan, and Ali Maatouk. 2024. Telco-RAG: Navigating the Challenges of Retrieval-Augmented Language Models for Telecommunications. arXiv:cs.IR/2404.15939

[6] Tilmann Bruckhaus. 2024. RAG Does Not Work for Enterprises. arXiv:cs.SE/2406.04369

[7] Patrice Béchard and Orlando Marquez Ayala. 2024. Reducing hallucination in structured outputs via Retrieval-Augmented Generation. arXiv:cs.LG/2404.08189 https://arxiv.org/abs/2404.08189

[8] Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri.

2024. The Power of Noise: Redefining Retrieval for RAG Systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24)*. ACM, 719–729.

[9] Jialin Dong, Bahare Fatemi, Bryan Perozzi, Lin F. Yang, and Anton Tsitsulin. 2024. Don't Forget to Connect! Improving RAG with Graph-based Reranking. arXiv:cs.CL/2405.18414

[10] Yi Dong, Ronghui Mu, Gaojie Jin, Yi Qi, Jinwei Hu, Xingyu Zhao, Jie Meng, Wenjie Ruan, and Xiaowei Huang. 2024. Building guardrails for large language models. *arXiv preprint arXiv:2402.01822* (2024).

[11] Bernardo Hernandez-Salinas, Juan Terven, E.A. ChaveZ-Urbiola, Diana M. Cordova-Esparza, Julio A. Romero-Gonzalez, Amadeo Arguelles, and Ilse Cervantes. 2024. IDAS: Intelligent Driving Assistance System using RAG. *IEEE Open Journal of Vehicular Technology* (2024), 1–27.

[12] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232* (2023).

[13] Xinke Jiang, Yue Fang, Rihong Qiu, Haoyu Zhang, Yongxin Xu, Hao Chen, Wentao Zhang, Ruizhe Zhang, Yuchen Fang, Xu Chu, Junfeng Zhao, and Yasha Wang. 2024. TC-RAG:Turing-Complete RAG's Case study on Medical LLM Systems. arXiv:cs.IR/2408.09199

[14] Ziyan Jiang, Xueguang Ma, and Wenhu Chen. 2024. LongRAG: Enhancing Retrieval-Augmented Generation with Long-context LLMs. arXiv:cs.CL/2406.15319 https://arxiv.org/abs/2406.15319

[15] Bowen Jin, Jinsung Yoon, Jiawei Han, and Sercan O. Arik. 2024. Long-Context LLMs Meet RAG: Overcoming Challenges for Long Inputs in RAG. arXiv:cs.CL/2410.05983 https://arxiv.org/abs/2410.05983

[16] Soki Kawashima, Shun Shiramatsu, and Takeshi Mizumoto. 2024. Development of RAG System for Digital Transformation of Local Government and Considering Optimal Document-Segmentation Methods. In *Proc. of Ninth International Congress on Information and Communication Technology*. Springer Nature, 603–617.

[17] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:cs.CL/2005.11401 https://arxiv.org/abs/2005.11401

[18] Dongfang Li, Zetian Sun, Xinshuo Hu, Zhenyu Liu, Ziyang Chen, Baotian Hu, Aiguo Wu, and Min Zhang. 2023. A Survey of Large Language Models Attribution. arXiv:cs.CL/2311.03731 https://arxiv.org/abs/2311.03731

[19] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Ass. for Computational Linguistics, 74–81.

[20] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-Task Deep Neural Networks for Natural Language Understanding. In *ACL 2019*. https://www.microsoft.com/en-us/research/publication/multi-task-deep-neural-networks-for-natural-language-understanding-2/

[21] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[22] Nicholas Matsumoto, Jay Moran, Hyunjun Choi, Miguel E Hernandez, Mythreye Venkatesan, Paul Wang, and Jason H Moore. 2024. KRAGEN: a knowledge graph-enhanced RAG framework for biomedical problem solving using large language models. *Bioinformatics* 40, 6 (06 2024), btae353.

[23] Johny Moreira, Altigran da Silva, Edleno de Moura, and Leandro Marinho. 2024. A Study on Unsupervised Question and Answer Generation for Legal Information Retrieval and Precedents Understanding. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24)*. ACM, 2865–2869.

[24] Zooey Nguyen, Anthony Annunziata, Vinh Luong, Sang Dinh, Quynh Le, Anh Hai Ha, Chanh Le, Hong An Phan, Shruti Raghavan, and Christopher Nguyen. 2024. Enhancing Q&A with Domain-Specific Fine-Tuning and Iterative Reasoning: A Comparative Study. arXiv:cs.AI/2404.11792

[25] Marco Polignano, Pierpaolo Basile, and Giovanni Semeraro. 2024. Advanced Natural-based interaction for the ITAlian language: LLaMAntino-3-ANITA. arXiv:cs.CL/2405.07101

[26] Vatsal Raina and Mark Gales. 2024. Question-Based Retrieval using Atomic Units for Enterprise RAG. arXiv:cs.CL/2405.12363

[27] Stephen E. Robertson and Karen Spärck Jones. 1976. Relevance weighting of search terms. *J. Am. Soc. Inf. Sci.* 27 (1976), 129–146. https://api.semanticscholar.org/CorpusID:45186038

[28] Andrea Santilli and Emanuele Rodolà. 2023. Camoscio: an Italian Instruction-tuned LLaMA. arXiv:cs.CL/2307.16456

[29] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval Augmentation Reduces Hallucination in Conversation. arXiv:cs.CL/2104.07567 https://arxiv.org/abs/2104.07567

[30] Irene Siragusa and Roberto Pirrone. 2024. Unipa-GPT: Large Language Models for university-oriented QA in Italian. arXiv:cs.CL/2407.14246

[31] Nirmalie Wiratunga, Ramitha Abeyratne, Lasal Jayawardena, Kyle Martin, Stewart Massie, Ikechukwu Nkisi-Orji, Ruvan Weerasinghe, Anne Liret, and Bruno Fleisch. 2024. CBR-RAG: Case-Based Reasoning for Retrieval Augmented Generation in LLMs for Legal Question Answering. In *Case-Based Reasoning Research and Development*. Springer Nature Switzerland, Cham, 445–460.

[32] Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, and Zheng Li. 2024. Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Vol. 33. ACM, 2905–2909.

[33] Shenglai Zeng, Jiankun Zhang, Pengfei He, Yue Xing, Yiding Liu, Han Xu, Jie Ren, Shuaiqiang Wang, Dawei Yin, Yi Chang, and Jiliang Tang. 2024. The Good and The Bad: Exploring Privacy Issues in Retrieval-Augmented Generation (RAG). arXiv:cs.CR/2402.16893

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

[35] Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K. Qiu, and Lili Qiu. 2024. Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely. arXiv:cs.CL/2409.14924 https://arxiv.org/abs/2409.14924