TUM Data Innovation Lab
Munich Data Science Institute (MDSI)
Technical University of Munich

&

Data Reply DE

Final report of project:

Real-time Augmentation of LLM for Enhanced Document Knowledge Retrieval

Authors	Jonas Knupp,
	Promwat Guide Angsuratanawech,
	Praveen Mohandas
Mentor	Antonio Di Turi (Data Reply)
TUM Mentor	Dr. Alessandro Scagliotti
Project Lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (MDSI)

July 2024

Abstract

In the age of information and big data, people and companies face increasing challenges in managing and utilizing their large amount of data. Effective knowledge retrieval and processing broad data sources is difficult and lucrative. Employees usually invest a lot of time looking for relevant details from several sources and a vast amount of files, which leads to inefficiencies and potentially overlooked information.

This project aims to streamline this process by developing a unified, user-friendly platform using Retrieval-Augmented Generation (RAG) to enhance knowledge retrieval and assist in handling internal document collections. This has the potential to significantly increase productivity, reduce costs, and lead to more informed decision making.

Our project has a serverless backend architecture, being cost-efficient and scalable. Resembling a RAG pipeline, it consists of two main phases: ingestion and retrieval. During ingestion, raw data is preprocessed, chunked, converted to meaningful vectors, and saved in a vector database. During retrieval, a query is converted into a vector to identify relevant data chunks, which are then used by an LLM to generate a response. Our system can handle a variety of rich document formats, including PDF, PowerPoint, Excel, and Markdown. It facilitates a multi-modal pipeline, to handle textual as well as visual content well. It supports hierarchical chunking, table parsing, and more. Ingestion and retrieval leverage efficient algorithms like Hierarchical Navigable Small Worlds (HNSW). The frontend is a single-page application hosted on AWS Lambda, built with modern web technologies. The system is user-friendly, minimalistic, scalable, and cost-efficient.

Contents

1 Introduction

1.1 Motivation and Problem Definition

Effective retrieval and utilization of knowledge from vast data sources is crucial in the modern business environment. However, employees often spend significant time searching for relevant data across multiple sources, leading to inefficiencies and potentially overlooked information. This project aims to streamline this process by developing a unified, user-friendly platform using Retrieval-Augmented Generation (RAG) to enhance knowledge retrieval and assist in handling internal document collections. This has the potential to significantly increase productivity, reduce costs, and lead to more informed decision making.

1.2 Retrieval-Augmented Generation (RAG)

Large Language Models (LLMs) are capable of modeling natural language text by learning statistical patterns. They can produce human-like text, but the factuality of their output is not guaranteed, resulting in subtly or drastically wrong responses to potentially critical questions. Moreover, adjusting the internal knowledge of a generally pretrained LLM with domain-specific, often changing, and detailed information is challenging. Retrieval-Augmented Generation aims to address these issues by providing LLMs with contextually relevant information when needed, hence reducing the probability of wrong answers, allowing citation of trusted sources, and subsequently helping to build user trust.
RAG adds two phases before the LLM-based response generation, **ingestion** (section 2.1) and **retrieval** (section 2.2. During ingestion, data is extracted from files, further processed, chunked, and stored, such that during retrieval it can be efficiently searched through, in order to find relevant chunks for a given query. These chunks are then used as additional input for an LLM, to generate a response that is supported by the retrieved information.

1.3 RAG for Request for Proposal

A Request for Proposal (RfP) is an important document created by an organization to announce and outline the specifics of a project and invite bids from potential contractors or service providers. An RfP typically outlines the project background, scope of work, vendor qualifications, proposal guidelines, evaluation criteria, contract terms, and timeline. Analyzing RfPs and generating good proposal documents is crucial for potential providers to secure contracts.
Our industry partner for this project, Data Reply, is an international tech consultancy firm. Typically, they receive 1-2 RfPs per week from various organizations. Due to the intensive workload required to create a comprehensive proposal, the firm can only produce one proposal per month. This results in an average of 3-7 missed opportunities each month to compete in a bidding process, leading to lost business prospects. Being introduced to this opportunity for improvement while working on our project, we expanded the usage domain of our RAG chatbot from general Q&A to supporting employees in analyzing RfPs and drafting parts of proposals based on the company's internal documents.

1.4 AWS and Serverless Infrastructure

One objective of our project was to use Amazon Web Services (AWS). This includes using AWS Bedrock for LLMs and AWS RDS for the database, keeping data from Data Reply's clients in an already approved environment. Using AWS also makes it easier to later handover the project to Data Reply with its extensive expertise in AWS.
Moreover, we aimed for a mainly “serverless” infrastructure, i.e. AWS Lambda. A serverless computing service executes code in a customizable environment and automatically manages the underlying compute resources. Unlike alternative hosting approaches, it allows to run code without provisioning or managing servers. Continuous server operation is not required, and costs are mainly determined by the actually used compute time [4]. This makes serverless architectures particularly advantageous for applications with varying or long idle times. Hence, for a quite specialized chatbot service with initially unclear or relatively few requests per minute, this is an interesting direction to explore.

2 Backend

2.1 Ingestion

The ingestion involves extraction of meaningful data from various file types (e.g. PDF, Powerpoint, Excel, Markdown), as well as further processing and chunking to more manageable pieces of information, and finally converting chunks to meaningful numeric vectors (embeddings) for later retrieval based on semantic similarity to a query embedding.

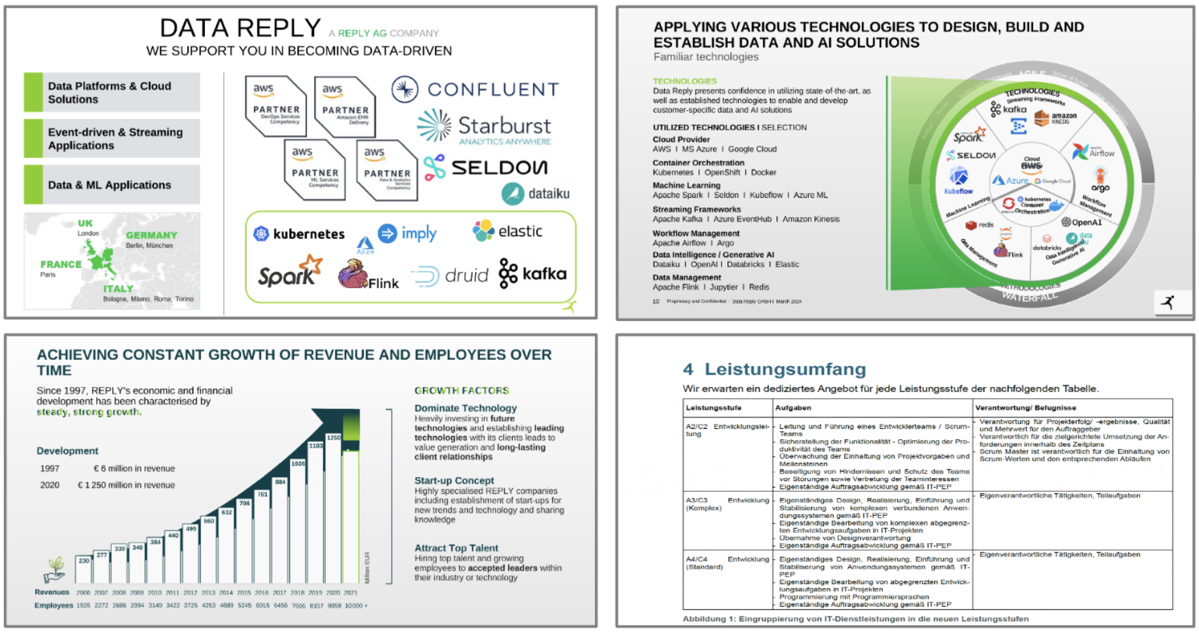
2.1.1 Rich Document Understanding

Data Reply has a diverse collection of documents, ranging from simple onboarding guides to visually rich presentations, as shown in fig. 1.
There are various approaches that try to extract meaningful content from documents containing complex layouts and relationships between text and visuals:

Dedicated layout understanding model: It aims to detect layouts in order to construct good semantic boundaries, reading order, etc. However, it is can be insufficiently accurate when dealing with versatile document layouts such as in presentations.

Convert images to text: By applying an image-to-text model, one can obtain textual descriptions of images, to be processed further by a text-only pipeline. However, this approach suffers from loosing details as well as the relationship between images and surrounding text. Especially for diagrams that consist of many small images and text elements, like often found in Powerpoint presentations and scientific papers, processing each element separately would be disastrous.

Convert all to images: A vision-centric pipeline that takes ßcreenshotsöf documents and then applies an image embedding model (and optionally optical character recognition (OCR)), is one of the most flexible approaches, but with current technology it can still result in inaccurate text detection, loosing text details in image embeddings, and slow performance.



DATA REPLY A REPLY AG COMPANY

WE SUPPORT YOU IN BECOMING DATA-DRIVEN

Data Platforms & Cloud Solutions

Event-driven & Streaming Applications

Data & ML Applications

UK
London
GERMANY
Berlin, München
FRANCE
Paris
ITALY
Bologna, Milano, Roma, Torino
aws PARTNER DevOps Services Competency
aws PARTNER Amazon EMR Delivery
aws PARTNER ML Services Competency
aws PARTNER Data & Analytics Services Competency
CONFLUENT
Starburst ANALYTICS ANYWHERE
SELDON
dataiku
kubernetes
Azure
imply
elastic
Spark
Flink
druid
kafka

APPLYING VARIOUS TECHNOLOGIES TO DESIGN, BUILD AND ESTABLISH DATA AND AI SOLUTIONS

Familiar technologies

TECHNOLOGIES

Data Reply presents confidence in utilizing state-of-the-art, as well as established technologies to enable and develop customer-specific data and AI solutions

UTILIZED TECHNOLOGIES I SELECTION

Cloud Provider
AWS I MS Azure I Google Cloud

Container Orchestration
Kubernetes I OpenShift I Docker

Machine Learning
Apache Spark I Seldon I Kubeflow I Azure ML

Streaming Frameworks
Apache Kafka I Azure EventHub I Amazon Kinesis

Workflow Management
Apache Airflow I Argo

Data Intelligence / Generative AI
Dataiku I OpenAI I Databricks I Elastic

Data Management
Apache Flink I Jupytier I Redis

10 Proprietary and Confidential - Data Reply GmbH I March 2024

ACHIEVING CONSTANT GROWTH OF REVENUE AND EMPLOYEES OVER TIME

Since 1997, REPLY's economic and financial development has been characterised by **steady, strong growth.**

Development

1997	€ 6 million in revenue
2020	€ 1 250 million in revenue

Revenues	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Million EUR	230	277	330	340	384	440	495	560	632	706	781	884	1036	1183	1250	
Employees	1925	2272	2686	2994	3149	3422	3725	4253	4689	5245	6015	6456	7606	8157	9059	10000 +

GROWTH FACTORS

Dominate Technology
Heavily investing in **future technologies** and establishing **leading technologies** with its clients leads to value generation and **long-lasting client relationships**

Start-up Concept
Highly specialised REPLY companies including establishment of start-ups for new trends and technology and sharing knowledge

Attract Top Talent
Hiring top talent and growing employees to **accepted leaders** within their industry or technology

4 Leistungsumfang

Wir erwarten ein dediziertes Angebot für jede Leistungsstufe der nachfolgenden Tabelle.

Leistungsstufe	Aufgaben	Verantwortung/ Befugnisse
A2/C2 Entwicklungsleitung	- Leitung und Führung eines Entwicklerteams / Scrum-Teams - Sicherstellung der Funktionalität - Optimierung der Produktivität des Teams - Überwachung der Einhaltung von Projektvorgaben und Meilensteinen - Beseitigung von Hindernissen und Schutz des Teams vor Störungen sowie Vertretung der Teaminteressen - Eigenständige Auftragsabwicklung gemäß IT-PEP	- Verantwortung für Projekterfolg/ -ergebnisse, Qualität und Mehrwert für den Auftraggeber - Verantwortlich für die zielgerichtete Umsetzung der Anforderungen innerhalb des Zeitplans - Scrum Master ist verantwortlich für die Einhaltung von Scrum-Werten und den entsprechenden Abläufen
A3/C3 Entwicklung (Komplex)	- Eigenständiges Design, Realisierung, Einführung und Stabilisierung von komplexen verbundenen Anwendungssystemen gemäß IT-PEP - Eigenständige Bearbeitung von komplexen abgegrenzten Entwicklungsaufgaben in IT-Projekten - Übernahme von Designverantwortung - Eigenständige Auftragsabwicklung gemäß IT-PEP	- Eigenverantwortliche Tätigkeiten, Teilaufgaben
A4/C4 Entwicklung (Standard)	- Eigenständiges Design, Realisierung, Einführung und Stabiliserung von Anwendungssystemen gemäß IT-PEP - Eigenständige Bearbeitung von abgegrenzten Entwicklungsaufgaben in IT-Projekten - Programmierung mit Programmiersprachen - Eigenständige Auftragsabwicklung gemäß IT-PEP	- Eigenverantwortliche Tätigkeiten, Teilaufgaben

Abbildung 1: Eingruppierung von IT-Dienstleistungen in die neuen Leistungsstufen

Figure 1: Example documents from Data Reply.

Hybrid of text and full-page images: Similar to the previous approach, no hand-crafted layout detection or image extraction is performed, and instead each page is treated as ßcreenshotïmage. But in addition to that, the computer-readable text inside a file is also extracted and processed. This hybrid pipeline is more complex to implement, but allows to benefit from both accurate text processing and overall contextual understanding of images and text together. However, the absence of smaller image chunks might still lead to loosing fine visual details in image embeddings.

We chose a hybrid text and image approach due to its mentioned benefits. This meant the need for a multi-modal pipeline (fig. 2). Image rendering was done by converting each file type to PDF (for easier rendering), relying e.g. on LibreOffice compiled inside a custom Docker image suitable for AWS Lambda, and then rendering each page. Text processing is more intricate and discussed next.

2.1.2 Text Extraction and Formatting

Challenges Extracting clean text from documents is not always straightforward, depending of the file type. Challenges include:

- Determining a good reading order.
- Formatting elements according to their semantic roles, e.g. headings, tables, etc.
- Excluding less relevant elements like headers, footers, and page numbers, while ensuring the main content remains intact.
- Detecting and merging text spans that belong together but are split in the file's raw data for various reasons.

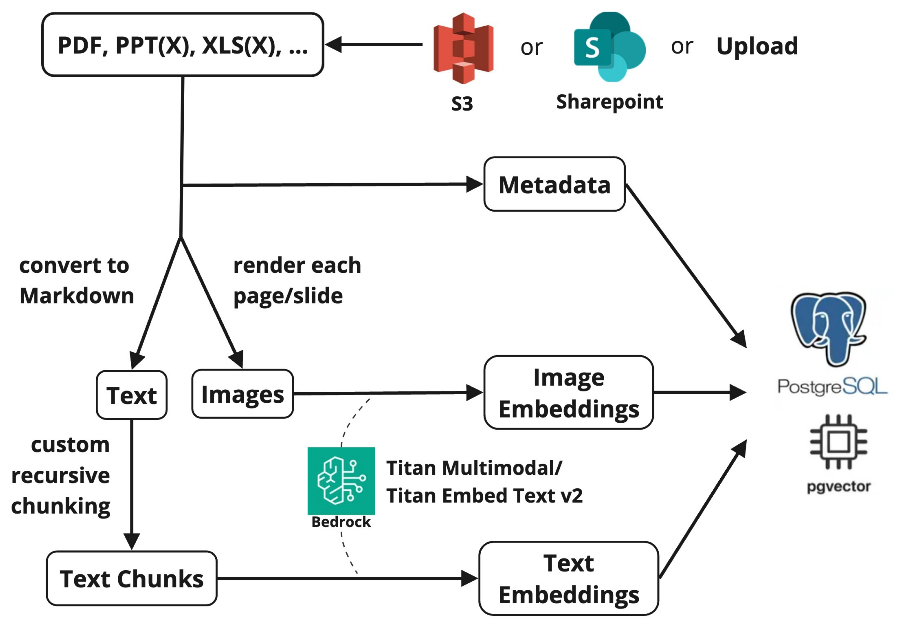


Figure 2: Multi-modal ingestion flow.

Milestone 1 We extract plain text from the converted PDFs using the *unstructured* library, using the reading order of the raw data in the file. This approach allows a unified pipeline for all file types convertible to PDF. Moreover, we determine the text length per page, to assign page numbers to each text chunk later, together with other file metadata.

Milestone 2 Using plain text often makes it hard to understand the semantic role of different text elements, e.g. headings of various levels, lists, tables, etc. Moreover, the *unstructured* library introduces heavy dependencies for some file types, making it slow to deploy to AWS Lambda. We therefore switched to manually using file-type-specific libraries to extract text. Doing so allows to better recover the reading order, using element coordinates, nesting, and metadata. For some file types like Powerpoint and PDF, headings of various levels are detected based on relative font sizes. Furthermore, we can format elements based on their role by using Markdown, being understood by most LLMs, e.g.:

```
# Title
## Subtitle

We can write **bold**, [links](https://example.com),
and much more, e.g. tables:

| Name  | Age |
| ----- | --- |
| Alice | 34  |
| Bob   | 32  |
```

Milestone 3 We further tuned the text extraction, e.g. allowing multi-line headings, merging split words, and compacting Markdown to reduce the resulting token count for LLMs.
We created separate text variants for embedding model and chat model. For the embeddings, our goal is to maximize the information density, so we dropped text styling like bold, italic, etc., and shortened URLs. For the chat model, i.e. the LLM receiving retrieved text, our goal is to keep all nuances, hence we retained full Markdown styling and URLs. This dual approach optimizes for each use case, enhancing both embeddings and post-retrieval processing.
Using only font sizes to detect headings their levels sometimes lead to wrong results. Such errors can propagate through all later content, resulting in wrong context information and thus lost retrieval accuracy. We therefore implemented a hybrid heading detection and alignment: Leveraging PDF metadata if available, we get an accurate table of contents (TOC). We then find the corresponding text elements in the file content that best fit the TOC, making use of coordinates provided in TOC metadata. This approach results in more accurate identification of section headings and their levels, which especially impacts retrieval accuracy for detailed text documents.

2.1.3 Text Chunking

Challenges The process of chunking is critical because embedding the entire content of a file at once is impractical as it can result in loss of details and does not facilitate the retrieval and citation of specific sections. The challenges of chunking include:

- Trading off between providing enough context to maintain meaningful and unambiguous chunks (→ larger chunks) and creating compact pieces of information that are fast to process, allow fine-grained citations, and can be compressed to fixed-size embeddings without loosing relevant details (→ smaller chunks).
- Recognizing semantic boundaries to split on, which can for instance involve determining sentence boundaries, including detection of abbreviations and solving other punctuation-related pitfalls.
- Avoid mistreating chunks out of context, which could otherwise induce false responses and bad retrieval accuracy.

Milestone 1 We started with chunking via a sliding window, which involves moving a fixed-size window over the text to create overlapping chunks. While this method is fast and captures some adjacent context, it does not take semantic boundaries into account, often splitting in the middle of paragraphs, sentences, or even words.

Milestone 2 We briefly considered semantic chunking via a language model, but due to performance and cost, we chose a classical software solution: After having text as Markdown, we parse it into an abstract syntax tree (AST), representing the hierarchy of elements like headings and paragraphs. We further parse paragraphs into sentences, words, and word-parts, using regular expressions. (We experimented with small language-model-based sentencizing, but the performance loss was not worth the added accuracy.)

The final AST is then split recursively at increasing depth until the chunks fit into a maximum chunk size. This keeps sentences, paragraphs, or even chapters intact whenever possible. Moreover, we include all headings of parent sections per chunk. This increases the contextual awareness, hence causes more meaningful embeddings, more accurate retrieval, and finally better response generation.

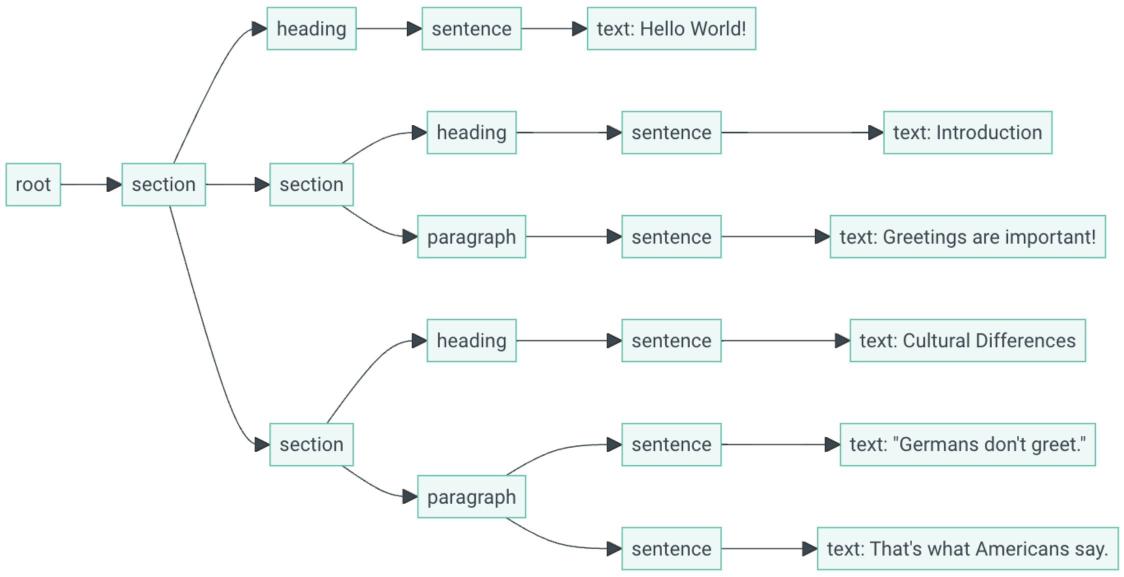


Figure 3: Simplified example of an AST.

Milestone 3 Building on our custom pipeline, we made further small adjustments and expanded the parsing to better support tables and lists. The latter resulted in also recursively chunking such nested structures, e.g. tables are first chunked by rows, then columns, then inner elements per cell, until fitting into the maximum chunk size. Crucially, We keep table headers for each chunk inside a table row/column/cell, similar to how we do it for section headings, to further profit from the mentioned benefits.

2.1.4 Text Embedding

Challenges Embeddings for texts and images are used to later perform semantic similarity search. Embedding-based retrieval provided advantages over classical text search, like being tolerant to synonyms, formulation variations, and spelling mistakes, and allowing e.g. to search for images by text if text and image vector spaces are aligned. However, it comes with own challenges and disadvantages, for instance:

- Embedding vectors have a fixed size (e.g. 512, 1024, or 2048 elements), with smaller vectors being faster to process while larger ones have more capacity.
- Embedding models must be well trained in order to capture meaning. Models for text have to understand the used languages (in our case: German and English), and models for images have to understand the provided visual elements (in our

case: document layouts, text of various fonts and languages, diagrams, etc.). For instance, image models trained only on photography are not suited for document understanding.

- Searching for verbatim text phrases is less likely to succeed with embeddings than with classical/exact text search.

Milestone 1 Due to our multi-modal approach, we chose the "*Titan Multimodal*" model to produce embeddings from texts or images in a common vector space, allowing to search for images by text, which is necessary for our use case. However, the significant issue is that it primarily supports English, therefore delivering an insufficient retrieval accuracy for our German content.

Milestone 2 For text retrieval, we switched to the multi-lingual modal "*Titan Embed Text v2*", allowing much more precise German text retrieval. Unfortunately, this model only supports text, leaving us with the suboptimal "*Titan Multimodal*" for images. While the image retrieval accuracy in milestone 1 was better than for text, this switched now. As reasons, we suspect not only the insufficient understanding of German for the image model, but also the presence of large amounts of content per image (typically one per page) rather than smaller, more manageable chunks, as well as the image model's additional training objective of considering visual aspects rather than only content-specific details. We used UMAP [5] as a non-linear dimensionality reduction method to visualize the text embeddings in 2D, as shown in fig. 4. One might consider the clusters of milestone 1 to be better separated, but investigating closed, we saw that focused too much on distinguishing by the high-level topic per file, while the milestone 2 embeddings clustered on more details rather than superficial aspects.

2.2 Retrieval and RAG

2.2.1 Challenges

Retrieval-Augmented Generation (RAG) is increasingly popular due to its already mentioned benefits, but it is not without challenges, e.g.:

- There is an inherent tradeoff between recall and precision during retrieval: **Recall** counts the useful retrieved chunks relative to all retrieved chunks, while **precision** counts the useful retrieved chunks relative to all useful chunks. Optimizing for high recall may lead to retrieving more irrelevant information, thus lowering precision, while optimizing for high precision may result in missing relevant information, lowering recall.

- Allowing the user and LLM to control the retrieval should be done in a convenient and error tolerant way.

- The retrieval and response performance can greatly affect usability. Ensuring that a RAG system operates efficiently and cost-effectively, while also providing high-quality results, can be challenging, especially with serverless infrastructure.

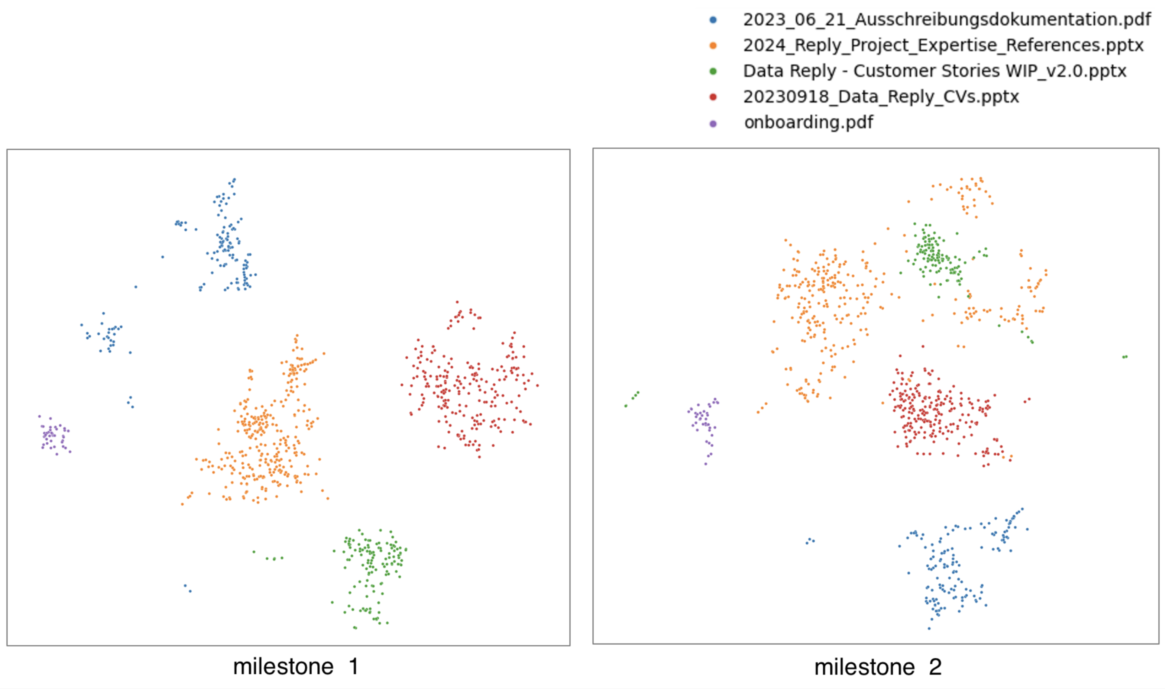


Figure 4: Embedding space visualization using UMAP. Milestone 1 seem to have better separated clusters, but milestone 2 has more semantically useful clusters. [5]

2.2.2 Hierarchical Navigable Small Worlds

Comparing a query embedding with every ingested embedding is too inefficient for large databases. Therefore, we leverage *Hierarchical Navigable Small Worlds (HNSW)* [3], an approximate nearest neighbor (ANN) search algorithm designed for dense high-dimensional vector spaces. HNSW provides high efficiency and precision with affordable performance, making it ideal for our document retrieval needs. The algorithm operates by constructing a hierarchy of neighborhood graphs from embeddings during ingestion. In the multi-layered graph, each layer represents a different level of abstraction. The lower layers connect nodes that are closer together, while the higher layers have sparser connections linking more distant nodes. This hierarchical approach allows for efficient navigation and search within large datasets by quickly narrowing down the search area through the layers, providing a good balance between speed and accuracy in finding approximate nearest neighbors.
As similarity measure we use negative dot-product. The embeddings are normalized beforehand, so this is equivalent to cosine similarity. Normalizing during ingestion and using just dot-product during retrieval further improves the retrieval performance.

2.2.3 Classical RAG

The classical approach to retrieval in RAG systems involves a hard-coded retrieval process before answer generation. This process is based on the original query or a reformulated query by a language model (LLM) to incorporate context from previous messages, aiming to create a standalone query. However, this approach is not very flexible because it relies on pre-determined rules and does not adapt dynamically to the needs of different

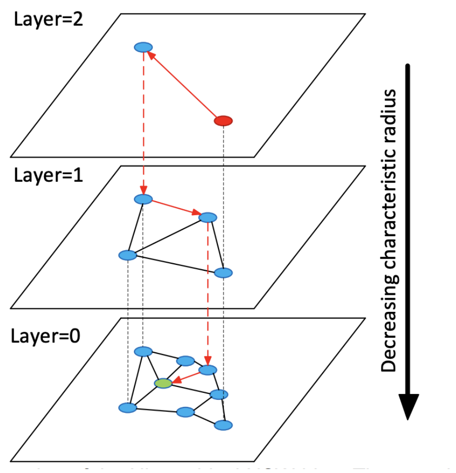


Figure 5: Rough diagram of Hierarchial Navigable Small Worlds (HNSW).

situations in conversational, e.g. allowing multiple or no retrieval steps per message.

2.2.4 Multi-Participant Messaging

Instead of a fixed pipeline like "query-retrieve-response" per user message, we decided to treat the user, AI, and knowledge base each as dedicated chat participant. This provides the flexibility to integrate various services on demand in the future, orchestrated by the AI. From the perspective of the AI, this is equivalent to tool-use or function-calling, but for the user interface, we abstract it be more like a multi-participant chat.
Some models like OpenAI's GPT and Anthropic's Claude 3 support tool-use natively. However, different model providers support different APIs and functionalities, and dispatching to tools is still the job of the client. We use cross-compatible adapters from *LangChain* [2] where possible. On top of that, the tool-use is abstracted into a a custom, small, and generic concept of multi-participant messaging. It allows arbitrary participants, full control over execution, and is designed with observability in mind, being asynchronous and interoperable by middlewares, used e.g. to limit the number of retrieval requests per user message.

Benefits:

- Full control by LLM, allowing flexible query reformulation. (See e.g. fig. 6.)
- Ability to avoid retrieval if history contains sufficient information, leading to faster answers. (See e.g. fig. 6.)
- Ability to query multiple times for complex subtask orchestration (although LLMs usually need good prompting for this to work, or manually implemented subtask execution concepts, which we do not have done yet).
- Using native tool-use/function-calling allows to reliably guide the LLM to adhere to a specified argument schema. This can be used to introduce additional retrieval parameters instead of just using a text query. We support e.g. filtering by files,

page range, etc. (See e.g. fig. 7.) There is future potential for controlling sorting, grouping, etc.

- Allows extensions with other tools/APIs, e.g. to add various capabilities in the future.

Drawbacks:

- More control by LLM means more opportunity for mistakes. Initially, we had routing of LLM messages to the user or knowledge base based on custom recipient syntax (like *[to ¡recipient¿]*), which caused the LLM to sometimes confuse user and knowledge base. Switching to native tool-use/function-calling APIs eliminated this weakness since some LLMs are trained for this, but it restricted our choice of LLMs to such. We therefore rely on *Anthropic Claude 3/3.5*.

- Letting the LLM invoke retrieval introduces an additional roundtrip, thus increasing the response time.

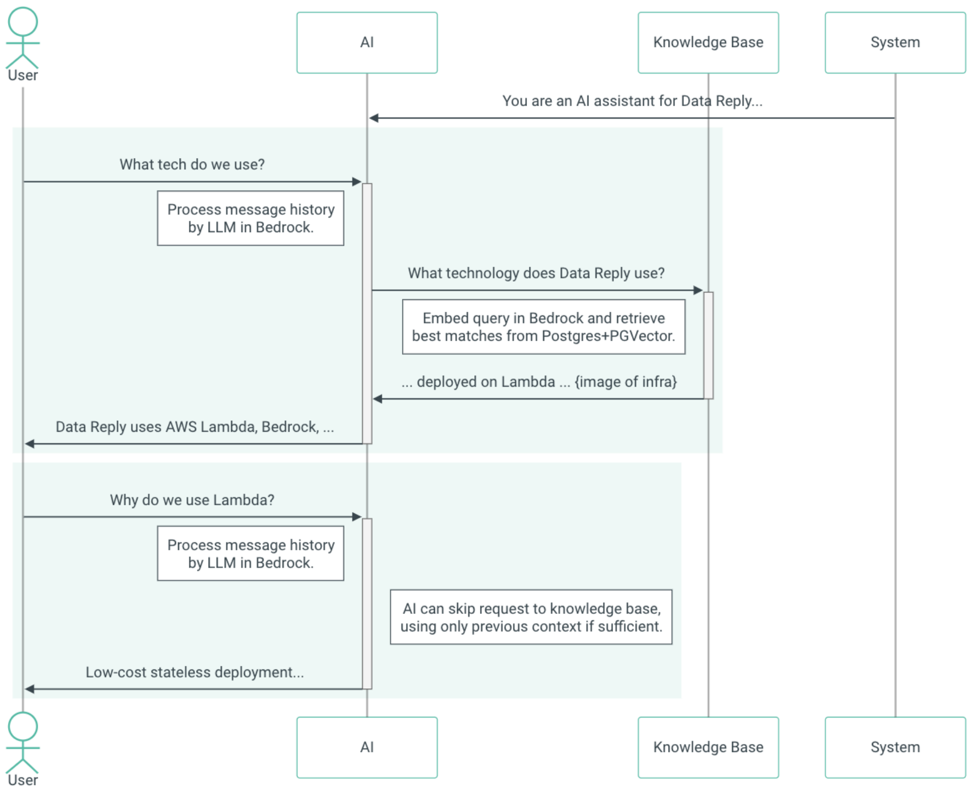

User
AI
Knowledge Base
System
You are an AI assistant for Data Reply...
What tech do we use?
Process message history by LLM in Bedrock.
What technology does Data Reply use?
Embed query in Bedrock and retrieve best matches from Postgres+PGVector.
... deployed on Lambda ... {image of infra}
Data Reply uses AWS Lambda, Bedrock, ...
Why do we use Lambda?
Process message history by LLM in Bedrock.
AI can skip request to knowledge base, using only previous context if sufficient.
Low-cost stateless deployment...
User
AI
Knowledge Base
System

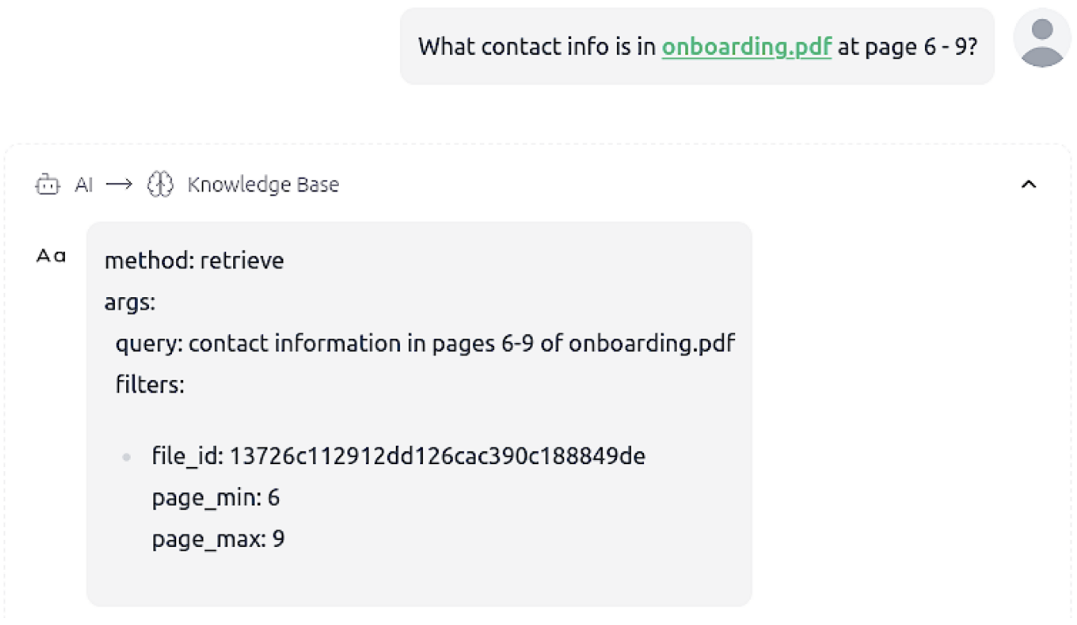
Figure 6: Example of message flow in multi-participant messaging with knowledge base retrieval as LLM-controlled tool.

We apply various techniques to improve and extend the RAG functionality, for instance:

Message time: Every message from user to LLM includes a prefix with the date/time it was send, which allows the LLM to correctly convert relative dates/times to absolute ones in order to produce standalone queries for the knowledge base retrieval.

Chunk metadata: We include metadata like chunk ID, chunk index, file ID/name, page number, etc. in each retrieved chunk. Together with the retrieval parameters, the LLM can invoke followup retrieval requests that expand the chunk context. In most cases however, the LLM currently does not leverage this opportunity, hinting at room for improving the system prompt.

Citations: The LLM is instructed to accurately cite retrieved chunks for generated claims by using the chunk ID to create a Markdown link. Clicking this link in the frontend opens a chunk preview, allowing the user to check and explore sources. This process is still not perfectly reliable, occasionally generating wrong citations if many chunks were retrieved.


What contact info is in onboarding.pdf at page 6 - 9?

```
method: retrieve
args:
 query: contact information in pages 6-9 of onboarding.pdf
 filters:

   • file_id: 13726c112912dd126cac390c188849de
     page_min: 6
     page_max: 9
```

Figure 7: Example of query filtering

Reranking: After retrieving text chunks based on embedding similarity, an efficient cross-encoder model can be applied to each chunk together with the query, in order to obtain more precise scores to further sort and filter chunks by relevance. This reranking allows to initially retrieve a larger number of chunks while keeping the workload for the more resource-intensive and slower chat LLM low. In contrast to the LLM, the reranking cross-encoder does not look at all chunks at once, hence reducing the issue of full attention having quadratic complexity. Moreover, reranking can be naively parallelized across all chunks. However, since our project was restricted to only using AWS, and AWS does not provide a reranking model, we had to host a small reranker via ONNX inside AWS

Lambda ourself. This resulted in slow performance, so we decided to disable it and instead focus on high-quality ingestion and retrieval.

2.3 Architecture

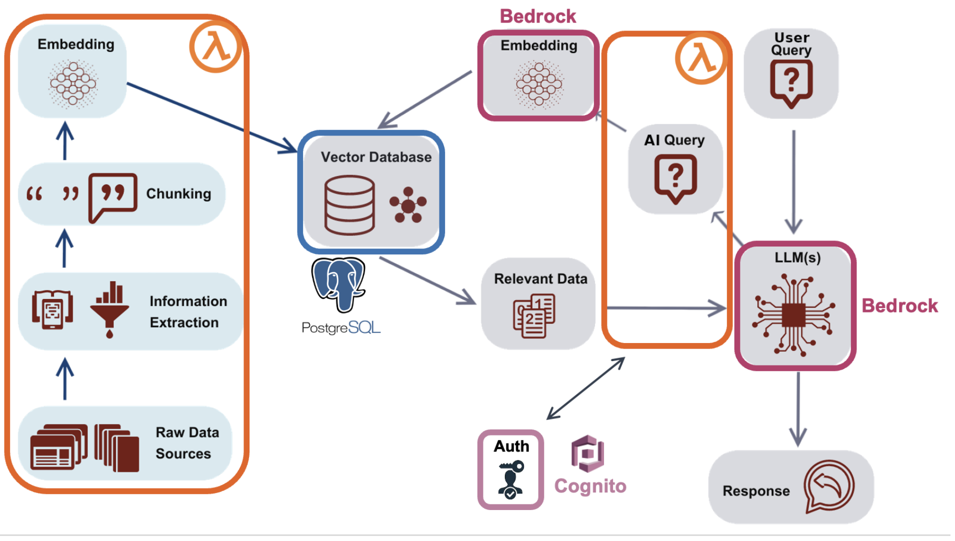


Figure 8: Infrastructure and high-level dataflow of the backend. Image adapted from [1].

Tech Stack:

AWS CloudFormation: Manages infrastructure as code, enabling automated provisioning and configuration of AWS resources, ensuring consistency and ease of management.

AWS Lambda: Handles serverless compute operations, executing the complete backend logic using Python without the need to manage servers, ensuring scalability and cost efficiency.

AWS API Gateway: Manages and exposes (RESTful) APIs, facilitating secure and scalable communication between the frontend and backend.

AWS RDS (Relational Database Service): Manages the database with PostgreSQL, configured with *pgvector* to provide a vector database for storing embeddings.

AWS Bedrock: Hosts LLMs such as Claude 3 for generating responses and Amazon Titan for embedding texts and images, ensuring fast AI inference.

3 Frontend

3.1 Overview

The goal of this project is to create a web chatbot for internal use at Data Reply. Therefore, one of the most important elements of this project is the frontend. It is undeniable that no matter how good the backend is, without a good user interface, it will not satisfy user experience. It is the first thing that users see when they interact with the system. Therefore, it is crucial to design the frontend to be as user-friendly as possible. We want to design the frontend to be as minimalistic as possible while allowing users to access every feature with minimal effort. Our goal is for users to be able to use the system without any introduction. Moreover, we want to provide flexibility for future feature additions. There are two hosted URLs: `https://chat.datareply.de/`, which is the production branch connected to the tested API and provides only stable features, and `https://chat-dev.datareply.de`, which is the development branch that allows developers to test new, untested features.

3.2 Tech Stack

The frontend is a single-page application hosted on AWS Lambda since we already use the AWS stack. Here is the list of components we use to create the frontend web application:

React: A JavaScript library for building user interfaces. It allows developers to create reusable UI components, making the development process more efficient and easier to manage.

Vite: A modern frontend build tool that offers fast server start, hot module replacement (HMR), and optimized builds, making development quicker and more efficient. This tool helps develop and build the source of the web application before it can be hosted on AWS Lambda. When choosing a build tool, we considered Vite and Next. We chose Vite since most of our functionality suits client-side rendering, not server-side rendering (which Next is known for), and we don't need the backend functionality from Next.

React Query: A powerful data-fetching library that simplifies fetching, caching, and synchronizing data. It makes handling the data fetching lifecycle-loading, data, and error-easier.

Amazon Cognito: Provides authentication, authorization, and user management for the web app. It stores separate chat histories for different users and ensures only authenticated users can use the system.

Microsoft Azure: Allows connection to SharePoint, which is the main storage for Data Reply employees. They can ingest files from their drive into our system and reference them.

shadcn/ui: Prewritten UI components using Radix UI and TailwindCSS, allowing flexible configuration based on our needs.

React Mentions Lets users use '@' to mention files inline with text in a textarea, indicating which files the AI should focus on.

React Dropzone: Enables users to drag and drop files to upload to S3 buckets, which can be used for asking questions.

React Markdown: Renders the response message in Markdown format, with different text styles such as Heading 1, Heading 2, Body, List, etc., making it easy for users to follow the response.

React Photo View: Allows users to scroll through selected images from files attached to messages.

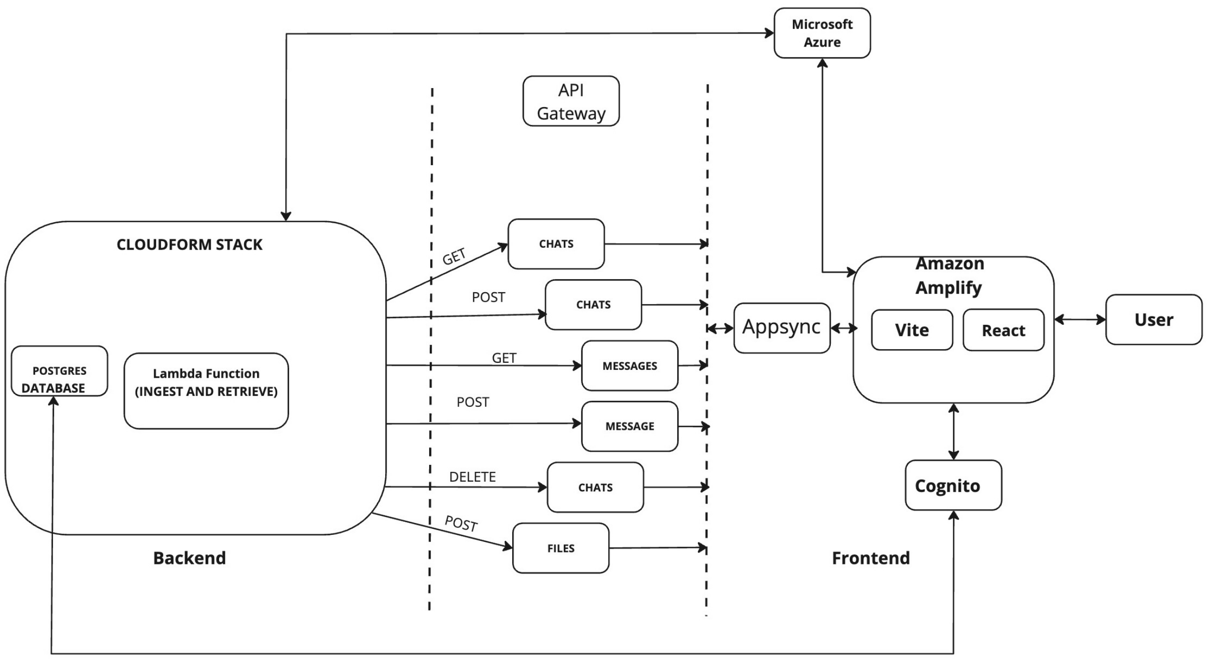


Figure 9: Frontend Architecture Diagram

3.3 Components Structure

The frontend is divided into several components, each with its own responsibility. Here is the structure of the components: The `App.` layer is the main component that contains the entire frontend application. Underneath it are `AuthContext`, which communicates with `AWS Cognito` to handle authentication, and `React Query`, which interacts with our backend and provides React Hooks for easily fetching data in the child components.
Inside the `FileDialog` component, we have `UploadFileDialog`, which makes an HTTP request to our backend to upload files to S3 buckets and ingest them into our database. Similarly, `SharepointDialog` contains `ListSharepointFiles`, which fetches files from Microsoft Azure and displays them in the frontend, allowing users to choose which files to ingest into our system via `IngestionConfirmDialog`.

3.4 Progress

Milestone 1 The project was handed over from Data Reply with initialized frontend code using React, but it didn't meet our needs. We started fresh with a new React app using Vite. For this milestone, we built core functionality, setting up the app with

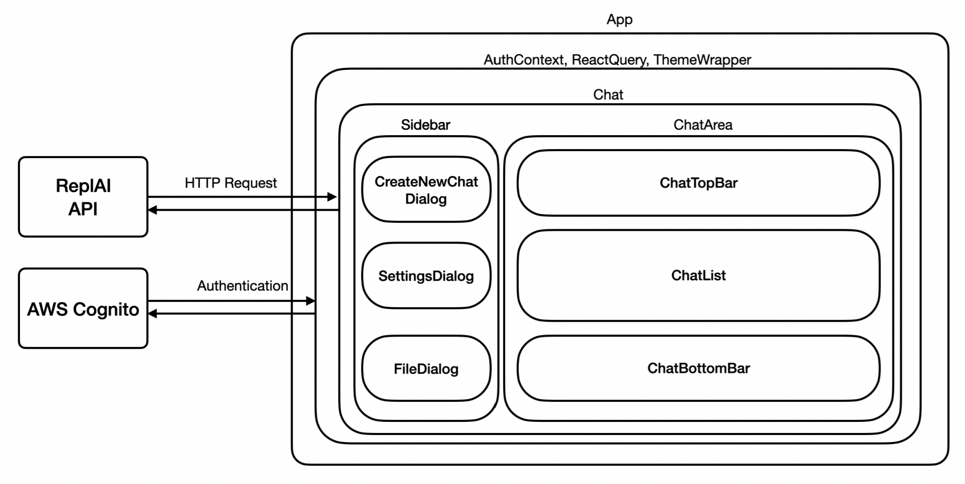


Figure 10: Component Diagram of App layer

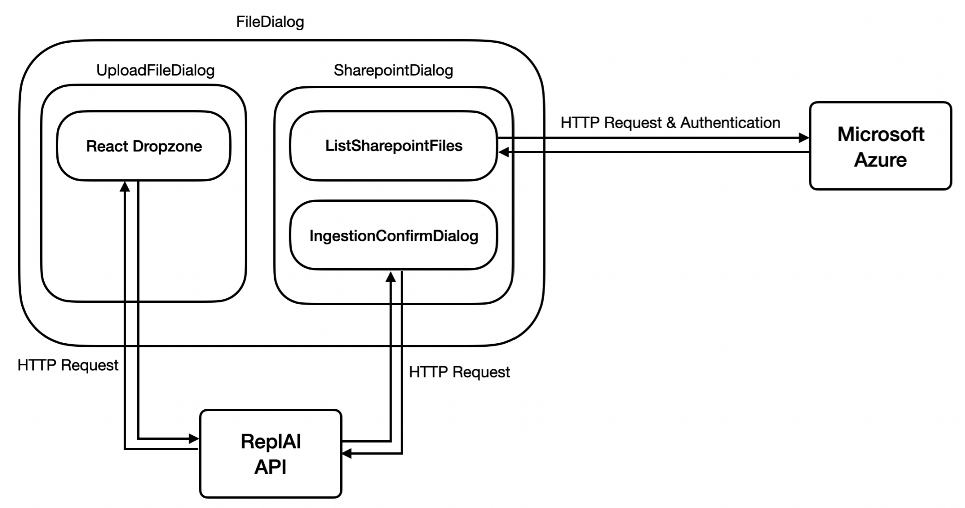


Figure 11: Component Diagram of File Dialog layer

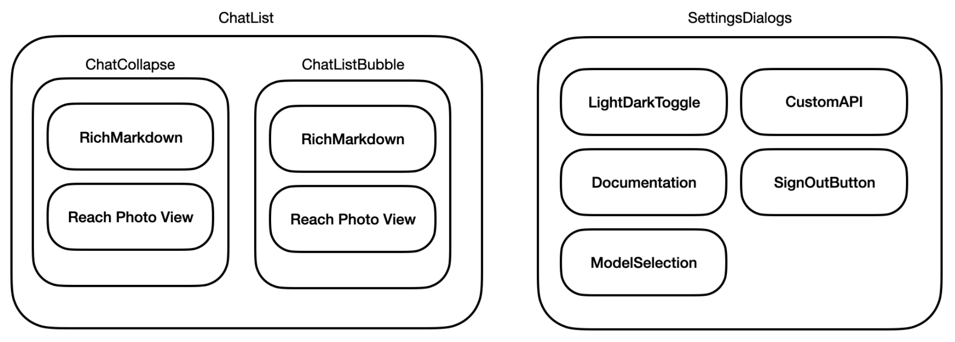
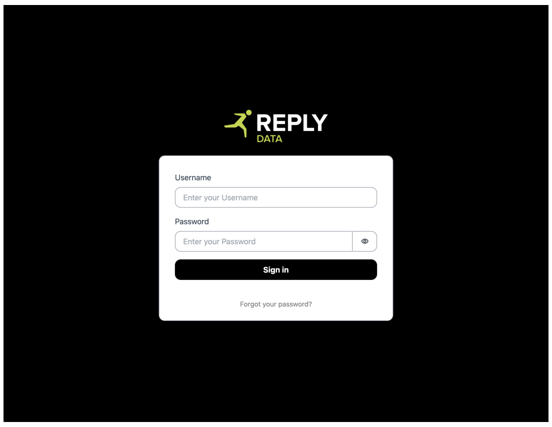
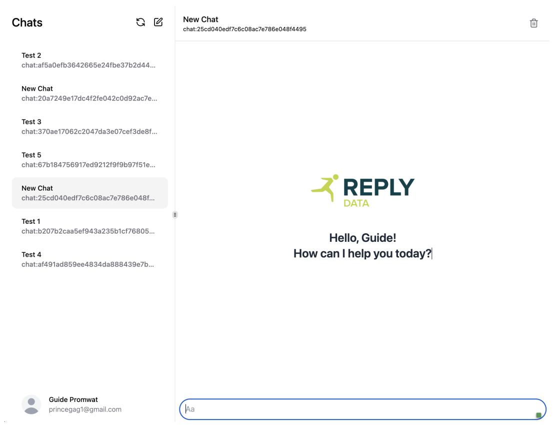


Figure 12: Component Diagram of SettingsDialog and ChatList

essential libraries like React Query, shadcn/ui, and TailwindCSS. We created a login page using Amazon Cognito and AWS Lambda UI components. We also initialized sidebar and chat area components for testing API interactions, including creating, sending, and deleting chats. Error handling and loading indicators were implemented. We added a collapsible section in the chat area to show how the final answer was derived, and included functionality to select custom APIs for testing different backend versions.



(a) Login Page


Chats
Test 2
chat:af5a0efb3642665e24fbe37b2d44...
New Chat
chat:20a7249e17dc4f2fe042c0d92ac7e...
Test 3
chat:370ae17062c2047da3e07cef3de8f...
Test 5
chat:67b184756917ed9212f9f9b97f51e...
New Chat
chat:25cd040edf7c6c08ac7e786e048f...
Test 1
chat:b207b2caa5ef943a235b1cf76805...
Test 4
chat:af491ad859ee4834da888439e7b...
Guide Promwat
princegag1@gmail.com
New Chat
chat:25cd040edf7c6c08ac7e786e048f4495
REPLY
DATA
Hello, Guide!
How can I help you today?
Aa

(b) Initalized Chat Page

Figure 13: Achievement from Milestone 1

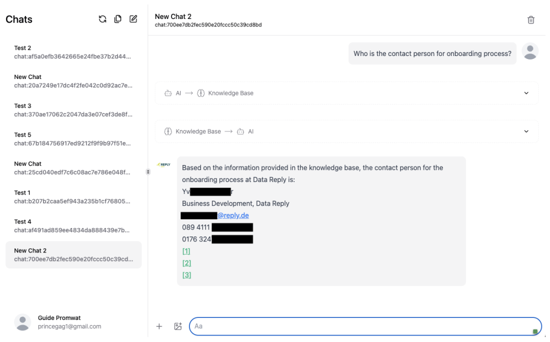
Milestone 2 After Milestone 1, we presented the app to Data Reply. They wanted it to generate RfP documents. Since the app was for Q&A style conversations, it wasn't suitable for long documents. Therefore, we focused on core features like image references, file ingestion from SharePoint, full-screen picture preview, and a centralized settings dialog with theme selection, documentation link, LLM model selection, and a customizable API.

Milestone 3 After Milestone 2, we focused on making the application's core functionality robust. We fixed bugs and added missing features. In Milestone 3, we added the ability to attach files inline with text sent to the AI, either by selecting a file or typing the file name with the '@' key. We also integrated file ingestion from Microsoft SharePoint, which was delayed from Milestone 2 due to authorization issues. Users must connect their Microsoft account to fetch and select files from SharePoint. Lastly, we configured the backend to respond in Markdown format for easier readability.

4 Results

4.1 Use Case: Q&A

Our final application is an internal chatbot for Q&A. Users log in to access the main chat page, where they see chat history on the left. They can start a new chat by clicking "New Chat", entering a name, and chatting with the AI. For example, when asked for the contact person of the onboarding document as shown in Figure 14a, the AI provided correct information, which can be verified via a citation link to review relavant text chunk as (Figure 14b)


Chats

Test 2
chat:af5a0efb3642665e24fbe37b2d44...

New Chat
chat:20a7249e17dc4f2fe042c0d92ac7e...

Test 3
chat:370ae17062c2047da3e07cef3de8f...

Test 5
chat:67b184756917ed9212f9f9b97f51e...

New Chat
chat:25cd040edf7c6c08ac7e786e048f...

Test 1
chat:b207b2caa5ef943a235b1cf76805...

Test 4
chat:af491ad859ee4834da888439e7b...

New Chat 2
chat:700ee7db2fec590e20fccc50c39cd...

Guide Promwat
princegag1@gmail.com

New Chat 2
chat:700ee7db2fec590e20fccc50c39cd8bd

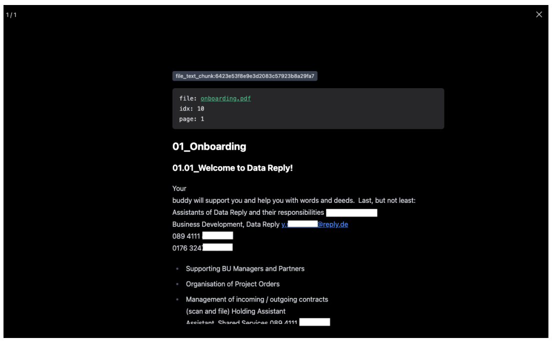
Who is the contact person for onboarding process?

AI → Knowledge Base

Knowledge Base → AI

Based on the information provided in the knowledge base, the contact person for the onboarding process at Data Reply is:
Yv[redacted]r
Business Development, Data Reply
[redacted]@reply.de
089 4111 [redacted]
0176 324[redacted]
[1]
[2]
[3]

(a) Asking AI for Contact Person


1 / 1

file_text_chunk:6423e53f8e9e3d2083c57923b8a29fa7

```
file: onboarding.pdf
idx: 10
page: 1
```

01_Onboarding

01.01_Welcome to Data Reply!

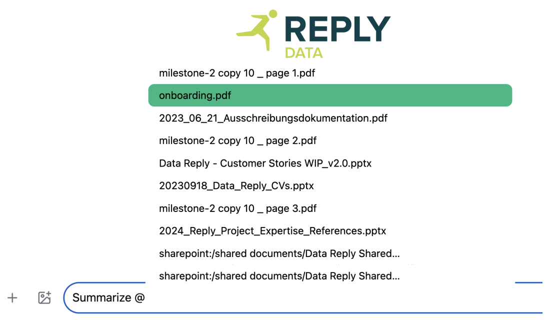
Your
buddy will support you and help you with words and deeds. Last, but not least:
Assistants of Data Reply and their responsibilities [redacted]
Business Development, Data Reply y.[redacted]@reply.de
089 4111 [redacted]
0176 324[redacted]

- Supporting BU Managers and Partners
- Organisation of Project Orders
- Management of incoming / outgoing contracts
  (scan and file) Holding Assistant
  Assistant, Shared Services 089 4111 [redacted]

(b) View Relevant Text Chunk

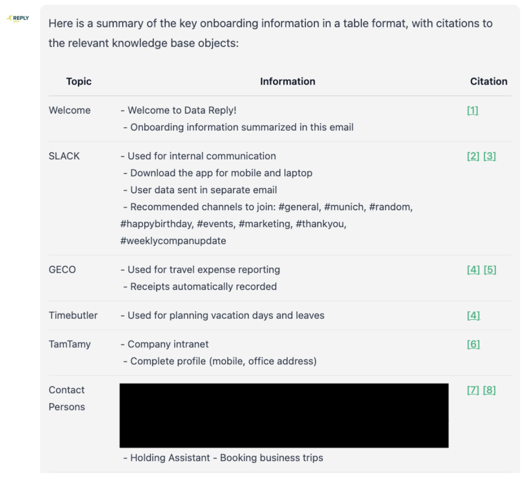
Figure 14: Example of Q&A Use Case with citation link

User can also mention files inline with the text by typing '@' followed by the file name (Figure 15a) or by clicking the plus button to open file dialog to select files. In this example, we ask for summary of the onboarding document which in this case we specific a file to focus on by mentioning it inline. Moreover, we also force the AI to reply back in a table format as shown in Figure 15b .


milestone-2 copy 10 _ page 1.pdf
onboarding.pdf
2023_06_21_Ausschreibungsdokumentation.pdf
milestone-2 copy 10 _ page 2.pdf
Data Reply - Customer Stories WIP_v2.0.pptx
20230918_Data_Reply_CVs.pptx
milestone-2 copy 10 _ page 3.pdf
2024_Reply_Project_Expertise_References.pptx
sharepoint:/shared documents/Data Reply Shared...
sharepoint:/shared documents/Data Reply Shared...

Summarize @

(a) Inline File Mention


Here is a summary of the key onboarding information in a table format, with citations to the relevant knowledge base objects:

Topic	Information	Citation
Welcome	- Welcome to Data Reply! - Onboarding information summarized in this email	[1]
SLACK	- Used for internal communication - Download the app for mobile and laptop - User data sent in separate email - Recommended channels to join: #general, #munich, #random, #happybirthday, #events, #marketing, #thankyou, #weeklycompanupdate	[2] [3]
GECO	- Used for travel expense reporting - Receipts automatically recorded	[4] [5]
Timebutler	- Used for planning vacation days and leaves	[4]
TamTamy	- Company intranet - Complete profile (mobile, office address)	[6]
Contact Persons	[redacted] - Holding Assistant - Booking business trips	[7] [8]

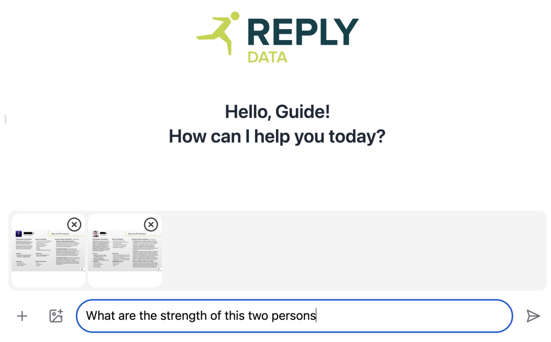
(b) Response from Inline File Mention

Figure 15: Example of Inline File Mention

To specify a more specific part of a file, the user can also pick images from files to the message by clicking the photo icon. In this case, we select two images from Data Reply's CV to ask for the strengths it found in both CVs as shown in Figure 16a and Figure 16b.

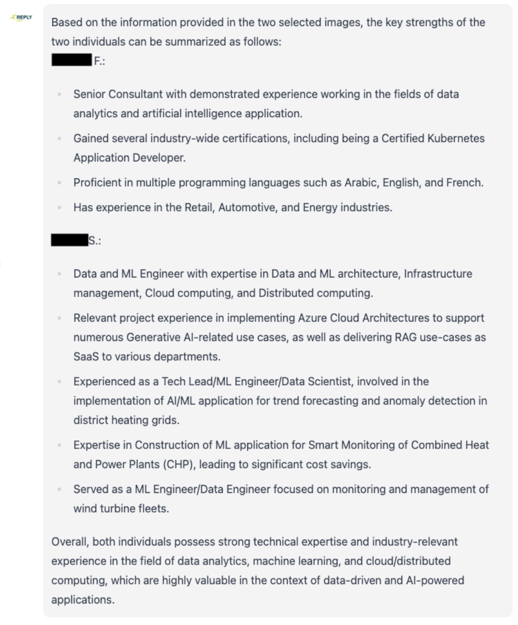
If retrieved from a database, the response will show what AI asked the Knowledge Base and what was found in the Knowledge Base. This is in a collapsible section that can be expanded to view details in a Markdown format, including headings, body text, and lists (Figure 17a). Users can also see a full-screen file preview by clicking the image (Figure 17b).

At the bottom of the chat list, the user can press their name to open the settings dialog. In the settings dialog, the user can change the theme, select the LLM model, and choose



What are the strength of this two persons

(a) Select Images from File



Based on the information provided in the two selected images, the key strengths of the two individuals can be summarized as follows:
[redacted] F.:

- Senior Consultant with demonstrated experience working in the fields of data analytics and artificial intelligence application.
- Gained several industry-wide certifications, including being a Certified Kubernetes Application Developer.
- Proficient in multiple programming languages such as Arabic, English, and French.
- Has experience in the Retail, Automotive, and Energy industries.

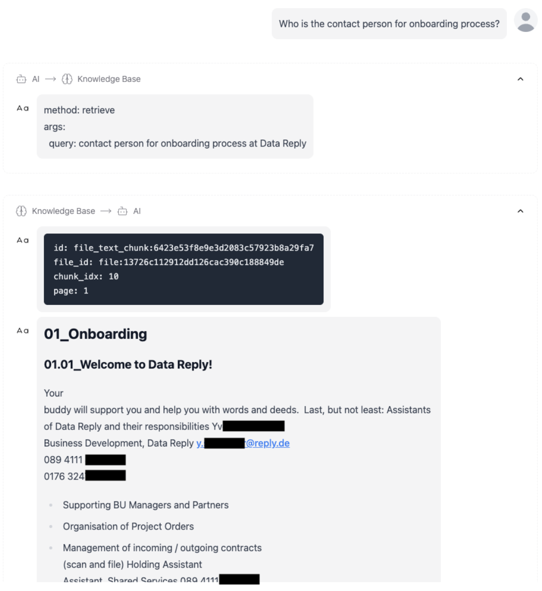
[redacted] S.:

- Data and ML Engineer with expertise in Data and ML architecture, Infrastructure management, Cloud computing, and Distributed computing.
- Relevant project experience in implementing Azure Cloud Architectures to support numerous Generative AI-related use cases, as well as delivering RAG use-cases as SaaS to various departments.
- Experienced as a Tech Lead/ML Engineer/Data Scientist, involved in the implementation of AI/ML application for trend forecasting and anomaly detection in district heating grids.
- Expertise in Construction of ML application for Smart Monitoring of Combined Heat and Power Plants (CHP), leading to significant cost savings.
- Served as a ML Engineer/Data Engineer focused on monitoring and management of wind turbine fleets.

Overall, both individuals possess strong technical expertise and industry-relevant experience in the field of data analytics, machine learning, and cloud/distributed computing, which are highly valuable in the context of data-driven and AI-powered applications.

(b) Response from Selected Images

Figure 16: Example of Selecting Images from File



Who is the contact person for onboarding process?

method: retrieve
args:
 query: contact person for onboarding process at Data Reply

```
id: file_text_chunk:6423e53f8e9e3d2083c57923b8a29fa7
file_id: file:13726c112912dd126cac390c188849de
chunk_idx: 10
page: 1
```

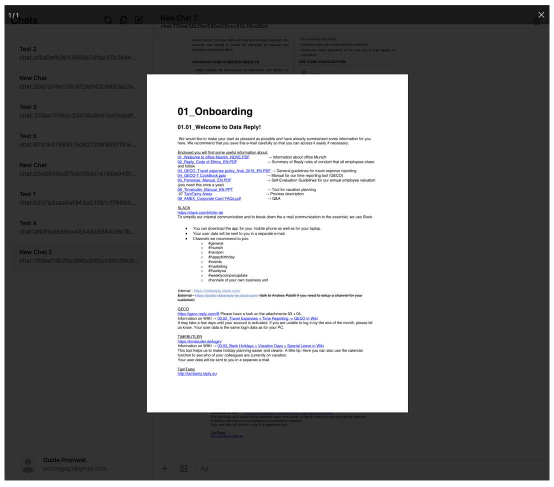
01_Onboarding

01.01_Welcome to Data Reply!

Your
buddy will support you and help you with words and deeds. Last, but not least: Assistants of Data Reply and their responsibilities Yv[redacted]
Business Development, Data Reply y.[redacted]@reply.de
089 4111 [redacted]
0176 324[redacted]

- Supporting BU Managers and Partners
- Organisation of Project Orders
- Management of incoming / outgoing contracts (scan and file) Holding Assistant
Assistant, Shared Services 089 4111[redacted]

(a) Expanded Response Details

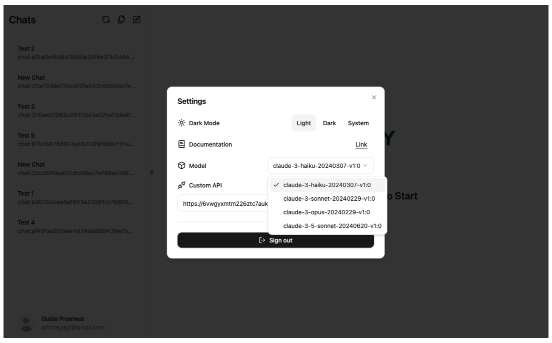


(b) File Preview from Image

Figure 17: Example of Expanded Response Details and File Preview

the API to use (Figure 18a). The user can also click on the Documentation button to open the documentation page (Figure 18b).

At the top of the chat list, there are Refresh, File, and Create New Chat buttons. The user can visit the file dialog by clicking the File button (Figure 19a). In the file dialog, the user can upload files to the system, preview uploaded files, and ingest files from Sharepoint as shown in Figure 19b.



Settings

Dark Mode: Light | Dark | System

Documentation: Link

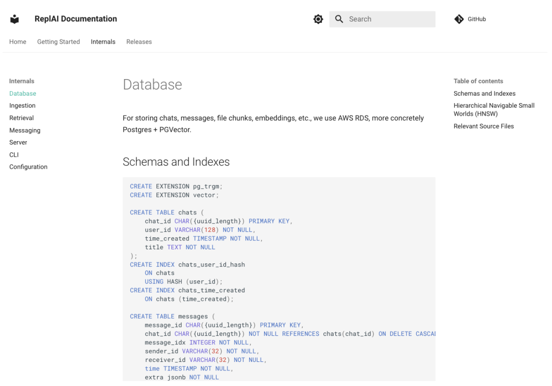
Model: claude-3-haiku-20240307-v1:0

- ✓ claude-3-haiku-20240307-v1:0
- claude-3-sonnet-20240229-v1:0
- claude-3-opus-20240229-v1:0
- claude-3-5-sonnet-20240620-v1:0

Custom API: https://6vwgyxmtm226ztc7auk

Sign out

Chats
Test 2
chat:af5a0efb3642665e24fbe37b2d44...
New Chat
chat:20a7249e17dc4f2fe042c0d92ac7e...
Test 3
chat:370ae17062c2047da3e07cef3de8f...
Test 5
chat:67b184756917ed9212f9f9b97f51e...
New Chat
chat:25cd040edf7c6c08ac7e786e048f...
Test 1
chat:b207b2caa5ef943a235b1cf76805...
Test 4
chat:af491ad859ee4834da888439e7b...
Guide Promwat
princegag1@gmail.com



ReplAI Documentation

Home | Getting Started | Internals | Releases

Database

For storing chats, messages, file chunks, embeddings, etc., we use AWS RDS, more concretely Postgres + PGVector.

Schemas and Indexes

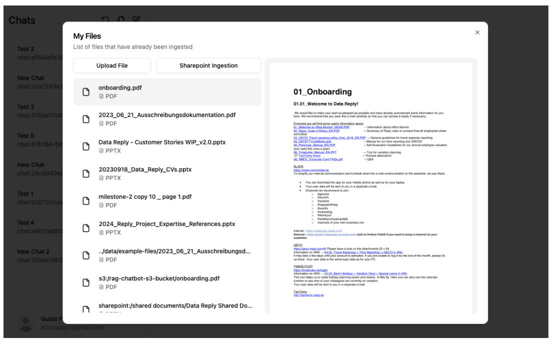
```
CREATE EXTENSION pg_trgm;
CREATE EXTENSION vector;

CREATE TABLE chats (
    chat_id CHAR({uuid_length}) PRIMARY KEY,
    user_id VARCHAR(128) NOT NULL,
    time_created TIMESTAMP NOT NULL,
    title TEXT NOT NULL
);
CREATE INDEX chats_user_id_hash
    ON chats
    USING HASH (user_id);
CREATE INDEX chats_time_created
    ON chats (time_created);

CREATE TABLE messages (
    message_id CHAR({uuid_length}) PRIMARY KEY,
    chat_id CHAR({uuid_length}) NOT NULL REFERENCES chats(chat_id) ON DELETE CASCAI
    message_idx INTEGER NOT NULL,
    sender_id VARCHAR(32) NOT NULL,
    receiver_id VARCHAR(32) NOT NULL,
    time TIMESTAMP NOT NULL,
    extra jsonb NOT NULL
```

(a) Settings Dialog (b) Documentation Page

Figure 18: Example of Settings Dialog and Documentation Page



My Files

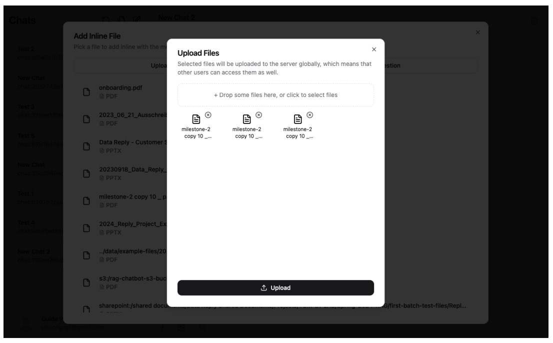
List of files that have already been ingested

Upload File | Sharepoint Ingestion

- onboarding.pdf — PDF
- 2023_06_21_Ausschreibungsdokumentation.pdf — PDF
- Data Reply - Customer Stories WIP_v2.0.pptx — PPTX
- 20230918_Data_Reply_CVs.pptx — PPTX
- milestone-2 copy 10 _ page 1.pdf — PDF
- 2024_Reply_Project_Expertise_References.pptx — PPTX
- ../data/example-files/2023_06_21_Ausschreibungsd... — PDF
- s3:/rag-chatbot-s3-bucket/onboarding.pdf — PDF
- sharepoint:/shared documents/Data Reply Shared Do...

01_Onboarding

01.01_Welcome to Data Reply!



Upload Files

Selected files will be uploaded to the server globally, which means that other users can access them as well.

+ Drop some files here, or click to select files

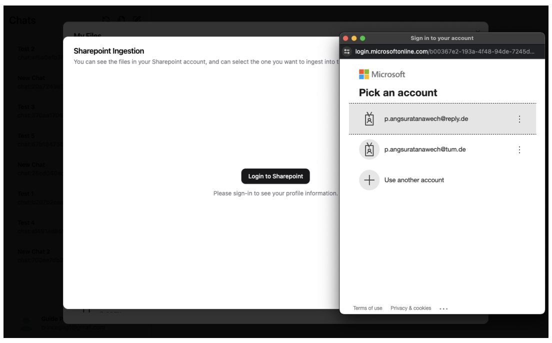
milestone-2 copy 10 _... | milestone-2 copy 10 _... | milestone-2 copy 10 _...

Upload

(a) File Dialog (b) Upload File Dialog

Figure 19: Example of File Dialog and Upload File Dialog

In order for the user to ingest files from Sharepoint, the user must first connect their Microsoft account to the system (Figure 20a).After that, the user can navigate to the folder and select the file to ingest (Figure 20b).



Sharepoint Ingestion

You can see the files in your Sharepoint account, and can select the one you want to ingest into t

Login to Sharepoint

Please sign-in to see your profile information.

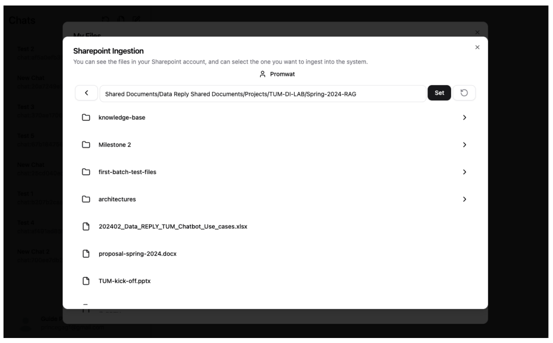
Sign in to your account

login.microsoftonline.com/b00367e2-193a-4f48-94de-7245d...

Microsoft

Pick an account

- p.angsuratanawech@reply.de
- p.angsuratanawech@tum.de
- Use another account



Sharepoint Ingestion

You can see the files in your Sharepoint account, and can select the one you want to ingest into the system.

Promwat

Shared Documents/Data Reply Shared Documents/Projects/TUM-DI-LAB/Spring-2024-RAG | Set

- knowledge-base
- Milestone 2
- first-batch-test-files
- architectures
- 202402_Data_REPLY_TUM_Chatbot_Use_cases.xlsx
- proposal-spring-2024.docx
- TUM-kick-off.pptx

(a) Sharepoint Login (b) Sharepoint File List

Figure 20: Example of Sharepoint Login and File List

Remark: Due to non-disclosure agreements, we censored some parts of the screenshots. The actual application is fully functional and does not contain any censored parts.

4.2 Use Case: Processing RfP Documents

The application isn't suitable to generate long documents, but can be used to ask questions about them. For example, we can ask "Regarding the `attachedFile`, list 5 Data Reply employees suitable for position 5.1." to obtain a list with short descriptions. It can't generate entire chapters due to backend and frontend limitations. To create full documents, we need more precise AI constraints, while current AI responses are too superficial.

5 Future Work

Since this project was limited to three months, it leaves much room for improvements, for instance:

Extended testing: Due to intricacies of real-world usage, we conducted qualitative evaluations by having users use the system and evaluate their responses. Our quantitative evaluation was very small-scale and should be extended to get reliable measurements. In a production environment, a rigorous testing pipeline should be implemented, not only for retrieval and response quality, but also including classical software unit tests, integration tests, performance tests, end-to-end tests, and more.

Hybrid retrieval: Searching for exact text, e.g. specific formulations, is better done with a classical text search, leveraging full-text indexing. State-of-the-art retrieval should therefore aim for a hybrid approach of using classical and embedding-based search.

System prompt tuning: We encountered that the LLM rarely invokes followup retrieval requests. This could likely be addressed to some extend by adjusting the system prompt. For models that are not trained for complex retrieval, injecting further instructions after each retrieval could lead to better behavior.

Subtask orchestration: By letting the LLM split complex tasks into subtasks, and focusing on each of them individually, might lead to better responses overall. For instance, applying a subworker per retrieved chunk can be seen as generalization of reranking, and allows a per-chunk action, e.g. discarding the chunk, recursing by issuing followup queries, or returning a summary to the parent worker.

Fine-grained knowledge extraction: Extracting triples (e.g. subject, predicate, object) from larger text/images chunks to build a detailed knowledge graph can lead to more precise relational retrieval. However, on our documents this approach tended to loose nuances of complex content. Moreover, it was challenging to unify synonyms for predicates/terms/formulations/etc. across processed document chunks. As alternative, and inspired by the attention mechanism of modern AI models, we experimented with generating just natural language question-answer pairs from documents, to be embedded and retrieved as usual. Our early tests seem promising.

Streaming responses: The more complex the work by the LLM gets, the longer is the waiting time for the user. A future development of our chat system should therefore implement the de-facto standard user experience of streaming responses/subtasks as they get generated.

File versioning: Reingesting files should not invalidate previous citations and conversations about older versions. We therefore see it as an important future work to have a file versioning system that adheres to immutability of history.

RfP document generation: Our current chat app is not well suited to generate and iteratively refine long documents like proposals to RfPs. Future development might improve this by providing a more collaborative editing experience and more control over what and how to generate.

6 Conclusions

This project has successfully developed a unified, user-friendly platform using Retrieval-Augmented Generation (RAG) to enhance knowledge retrieval and assist in handling internal document collections at Data Reply. The system has demonstrated several impressive capabilities:
The ingestion pipeline is highly capable, handling a variety of rich document formats including PDF, PowerPoint, Excel, and Markdown. It leverages efficient algorithms like Hierarchical Navigable Small Worlds (HNSW) to preprocess and index the content, enabling fast and accurate retrieval later on. The ingestion pipeline supports hierarchical chunking, table parsing, and other advanced techniques to extract meaningful content from complex documents.
The retrieval process is also highly flexible, allowing the Language Model (LLM) to dynamically control and refine the queries sent to the knowledge base. This multi-participant messaging approach provides great control and extensibility, enabling features like avoiding redundant retrieval if the chat history already contains sufficient information. The system also supports advanced retrieval parameters like filtering by file, page range, etc. to further narrow down the search.
The overall system architecture is designed to be serverless, cost-efficient and scalable, leveraging AWS services like Lambda, RDS, and Bedrock. This makes the platform very flexible and suitable for a specialized chatbot service with varying usage patterns. The frontend is a single-page application that provides a clean and intuitive user experience.
In summary, this project has delivered a powerful knowledge retrieval platform that can significantly improve productivity, reduce costs, and lead to more informed decision making at Data Reply. The system's advanced capabilities in document ingestion, retrieval, and response generation, combined with its efficient and scalable architecture, make it a valuable tool for the company.
As an additional proof of concept, we used our app to generate this conclusions chapter.

References

[1] *Best Practices in Retrieval-Augmented Generation.* URL: https://gradientflow.substack.com/p/best-practices-in-retrieval-augmented (visited on 07/19/2024).

[2] *LangChain: Use Cases and Tool Use.* URL: https://python.langchain.com/v0.1/docs/use_cases/tool_use/ (visited on 07/14/2024).

[3] Yury Malkov and Dmitry Yashunin. *Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs.* 2016. arXiv: 1603.09320 [cs.IR]. URL: https://arxiv.org/abs/1603.09320 (visited on 07/17/2024).

[4] *Serverless Retrieval-Augmented Generation on AWS.* URL: https://aws.amazon.com/startups/learn/serverless-retrieval-augmented-generation-on-aws (visited on 07/19/2024).

[5] *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.* URL: https://umap-learn.readthedocs.io/en/latest/ (visited on 07/19/2024).