A Large Language Model-based Web Application for Contextual Document Conversation

Asad Khan[1], Abdul Haseeb Malik[1], Talha Ahsan[1], Qazi Ejaz Ali[1]
[1]Affiliation Department of Computer Science, University of Peshawar, 25120, Pakistan.
*** Correspondence:** haseeb@uop.edu.pk

Citation | Khan. A, Malik. A. H, Ahsan. T, Ali. Q. E, "A Large Language Model based Web Application for Contextual Document Conversation", IJIST, Vol. 06 Issue. 04 pp 2069-2083, Dec 2024
DOI | https://doi.org/10.33411/IJIST/1147
Received | Nov 17, 2024 **Revised |** Dec 13, 2024 **Accepted |** Dec 15, 2024 **Published |** Dec 17, 2024.

The emergence of Large Language Models (LLM), such as ChatGPT, Gemini, and Claude has ushered in a new era of natural language processing, enabling rich textual interactions with computers. However, despite the capabilities of these new language models, they face significant challenges when queried on recent information or private data not included in the model's dataset. Retrieval Augmented Generation (RAG) overcame the problems mentioned earlier by augmenting user queries with relevant context from a user-provided document(s), thus grounding the model's response to inaccurate source material. In research, RAG enables users to engage interactively with their documents, instead of manually reading through their document(s). Users provide their document(s) to the system, which is then converted into vector indices, and used to inject contextual information into the user prompt during retrieval. The augmented prompt then enables the language model to contextually answer user queries. The research is composed of a web application, with an intuitive interface for interacting with the LIama 3.2 1B, an open-source LLM. Users can upload their document(s) and chat with the LLM in the context of their uploaded document(s).

Keywords: Application Learning, Natural Language Processing, AI Bots, Large Learning model, Contextual Document Conversation.

Large Language Model	LLM
Retrieval Augmented Generation	RAG
Natural Language Processing	NLP
Natural Language Understanding	NLU
Hypertext Transfer Protocol	HTTP
Hypertext Markup Language	HTM
Uniform Resource Locator	URL
Document Object Model	DOM
Application Programming Interface	API

Introduction:

Artificial Intelligence (AI) technology is utilized in NLP (expert systems) to allow computer applications to comprehend human language. Natural language processing is made up of two groups: language generator and understanding naturally. NLU refers to the process of interpreting or reading language, and language generator refers to the procedure of generating or writing. With natural language understanding and generation, large language processing integrates human language into machine learning, computational linguistics, and deep learning. Before natural language processing, inputs are processed in advance to get the data ready for classification. Text preprocessing is crucial before large language processing to avoid interference in textual analysis, as text often contains distractions and irrelevant content. Normalization, tokenization, noise and stop word removal, and lemmatization are common preprocessing text techniques. NLP has become widely used as this technology developed it began to be used in a variety of applications, such as sentiment analysis, which is the method of using computational techniques to recognize and classify the opinions conveyed in a piece of text, especially to determine the author's position (positive, negative, or neutral) regarding a particular topic, and speech recognition, which refers to the process of transforming spoken language into written text. ChatGPT is popular, especially among students, although it can finish tasks quickly it struggles with honesty and plagiarism [1]. Additionally, ChatGPT helps teachers create activities and course materials [2][3]. Research, amusement, code creation, clarification, and annotations, testing scenarios, pattern matching, documentation creation, code debugging, integration, transformation, and style, as well as a metaverse learning environment, are just a few of the opportunities that ChatGPT offers in addition to teaching[2][4][5][6].

Generative AI, especially Large Language Models (LLMs) have attained remarkable proficiency in generating human-quality text. These models are capable of performing tasks such as query response, summarization, translation, and text generation. These models are now starting to serve as alternatives to search engines, and traditional keyword-based searching for information retrieval[7]. However, these models encounter significant challenges, such as hallucination, and outmoded information. Retrieval Augmented Generation (RAG) tackles these issues by integrating outside knowledge sources[8].

A web-based application is being proposed that uses RAG to enable users to converse with an LLM in the context of their documents and gain relevant insights from their conversations. User comprehension of complicated documents can be improved by providing summaries and explanations of complex material. The application improves productivity, enhances the decision-making process, and enables deeper document understanding.

LLMs without an RAG subsystem do not incorporate the context of user-specific documents. These systems are also constrained by their knowledge cutoff date, rendering them ineffective for accessing the most current information [9][10]. Moreover, they have no awareness of private data as it was not part of the training dataset. While language models can enhance user comprehension, users should remain vigilant because these models can generate inaccuracies or false information [11]. Tying the model's responses to the context of user documents helps to reduce the likelihood of such inaccuracies [8].

The system converts user documents into text chunks calculates embedding for these text chunks and saves them in a vector database. When the user queries the system, using similarity search, the text chunks which are relevant to the user prompt are retrieved from the vector database. These text stacks are then passed into the context of the language model, along with the user query. The model uses the modified prompt to answer the user query in the context of the provided text chunks. The proposed solution is selected. After all, it is much more efficient than having to fine-tune an LLM for each user document because it requires extensive computation and is therefore unfeasible for this use case.

The ability of traditional find and search methods for finding information in a document is limited. We summarize our contributions as outlined below.:

- Improved efficiency by retrieving contextual information for the user query, and minimizing the user effort in reading a document.
- The accuracy of LLMs is increased by supplementing the model's response with relevant information, which might not be present in the training dataset of the model.
- The applicability of the LLMs is broadened by allowing the usage of private data, enabling users to manage their documents.
- To supplement model prompts with relevant document context.

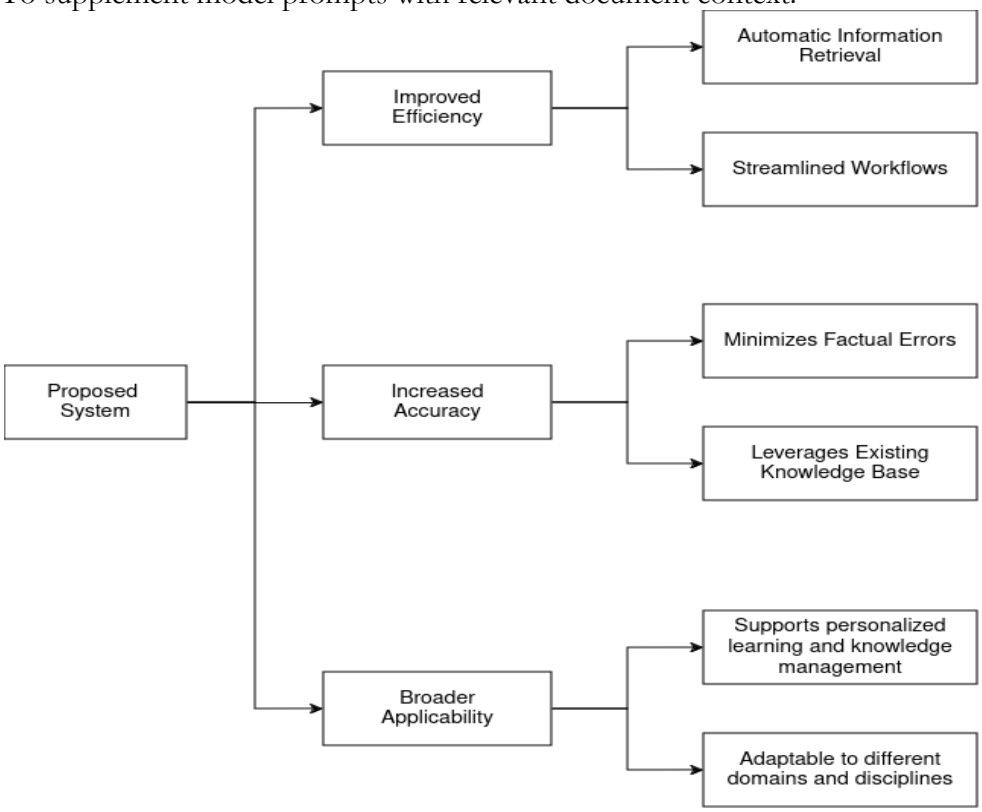


Figure 1: Proposed System

Existing Modules:

RIT, notably in the development of language models, and processing of natural language has revolutionized people's interaction with content based on text. Traditional approaches, such as PDF download and static reading require many tangential references for understanding or clarity. Additionally, this research emphasizes the necessity for an enhanced flexible and end user-focused strategy by integrating an engaging chat module that leverages the functionalities of a language model, specifically Meta's LLM. In our system, Llama 3.2 1B is the large language model that is implemented based on research on large language models like ChatGPT and GPT-3. These large language models showcase their strong capabilities in large language understanding (LLU), queries, summarizing text, and generating large language [12][13].

Existing systems, such as ChatGPT, lack the context of user documents. These systems are also limited because of their knowledge cutoff date, which makes them useless for the latest information [9][10]. Additionally, these systems are unaware of private data since it was not included in the dataset training model. Language models can improve user understanding, but the user must exercise caution since these models are prone to hallucinations [11]. Grounding the model's response to the user document context makes it less prone to hallucination [8].

Our research goal is to develop a chatbot powered by a large language model that assists users in navigating PDF documents more effectively. Research initiatives like " Chatbot as a

Broader Platform for Language earning: An Exploration and Summary of Chatbot Technology" demonstrate that it is possible to develop engaging chat interfaces for extracting tidings from PDFs. These researches highlight the necessity of understanding user needs, retrieving information from users, and employing techniques for generating LLM responses during conversational exchanges.

The absence of advanced artificial intelligence makes it difficult to comprehend and extract valuable data from PDF files because of variations in design, information encoding methods, and the possibility of including documents that have been scanned. Consequently, numerous recent studies have been conducted to investigate methods for text preprocessing, optical character recognition (OCR), and information retrieval techniques designed for various PDF formats. These research efforts are laying the groundwork for enhanced accuracy and precision in obtaining information for LLM interactions.

Large Language Models Comparison:

Based on the comparison in Table 1, the Claude language models are the most appropriate models for chatting with textual documents [14][15]. ChatGPT 3.5 is highly adaptable and can be used for almost any kind of activity. Its pricing is competitive. Moreover, its use in both personal and professional areas has commenced and is proliferating. On the other hand, the Claude LLM model is preferable when users have a high volume of prompts and wish to send chat prompts without circumnavigating. The maximum prompt size is one lack of tokens, allowing for approximately seventy-five thousand words to be included in a single prompt.

Table 1: Existing LLM models comparison

Model	API	MMLU	Bench Score	Parameters	Tasks	Cons
Microsoft T5	Pending	47.7	3.04	11 billion	Tunes Text Classification and Translation Fine custom	Use for commercial purposes is not allowed. Accessing it is difficult.
Google Palm	Yes	-	6.4	540 billion	Question-Answering summarization Text and Code Generation	Possibility of skewed or unsuitable results
Chat-3.5	Yes	70	7.94	175 billion	Text Generation Summarization Question Answering	Prejudices in Output Creation somewhat costly
Claude v1	Yes	75.6	7.9	-	Specific language tasks	Does not produce very human-like responses.

The goal of this research is to enhance understanding, accessibility, and information for a diverse audience by integrating immediate clarifications, environment-sensitive interpretations, and NLP features like translation and data extraction.

Methodology:

In this part, we outlined the structure and framework of our suggested large language model web application for contextual document conversation. The proposed research is composed of the following components:

Auth:

This component is responsible for authenticating users; the component requires the passport.js authentication middleware. It enables users to authenticate using their email and password or through authentication providers, such as Google or GitHub.

Chat:

This component enables users to upload their documents and chat with those documents using an LLM. The component also saves user chat history, allowing them to refer back to it if needed. The component requires two other components to work, the Auth component and the iOllama component, and uses them through their provided interfaces.

The iOllama required interface is provided by an external component iOllama, which facilitates running LLMs locally on a computer. iOllama provides a REST API for interacting with different LLMs.

User:

This component provides functionalities related to user management, such as blocking/deleting a user. This component is only utilized by admin users.

Feedback:

This component is responsible for storing and retrieving user feedback on chat messages. Only admins can access the provided feedback.

The Auth component is responsible for authentication and authorization. It is a guardrail against access by unauthorized users. Users must be authenticated through this component before they can proceed to interact with the other components. The Feedback component allows users to provide feedback to LLM responses in their chats. This feedback is visible to admins. The purpose of this component is to provide users with a way to provide feedback to LLM responses, so that the LLM may be further improved. Admin can use the User component for user management if they detect any unusual activity in the application they can block or delete user accounts.

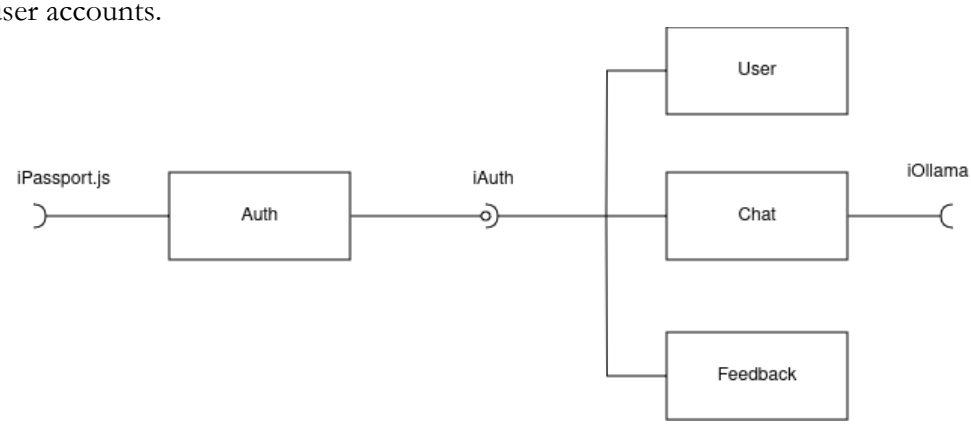


Figure 2: Main Components

Study Flow.

The flow of data through the application is illustrated in Figure 3. The diagram being shown is the level 0 in which the high-level movement of data through the application is shown. The end-user links with the application by providing a document, which is processed by the system and transformed into text chunks and stored along with their vector embeddings in the Vector DB process, which makes use of the vector index to retrieve relevant context from the index, and then calls the Generation process to respond to the user's query in terms of the given context. The Generation process also interacts with the MongoDB data-store, to persist the user query and the generated response. The Entity-Relationship diagram for the application is shown in Figure 4. The application includes five key business entities: Users, Chats, Messages, Feedback, and Categories. The relationship between the Users and Chat entities is classified as one-to-many, meaning a single user can have multiple Chats, while each Chat entity is associated

with only one User entity. There is a one-to-many relationship between the Chats and Messages entities. The Feedback and Messages entities maintain a one-to-one relationship. Lastly, the relationship between the Categories and Feedback entities is one-to-many.

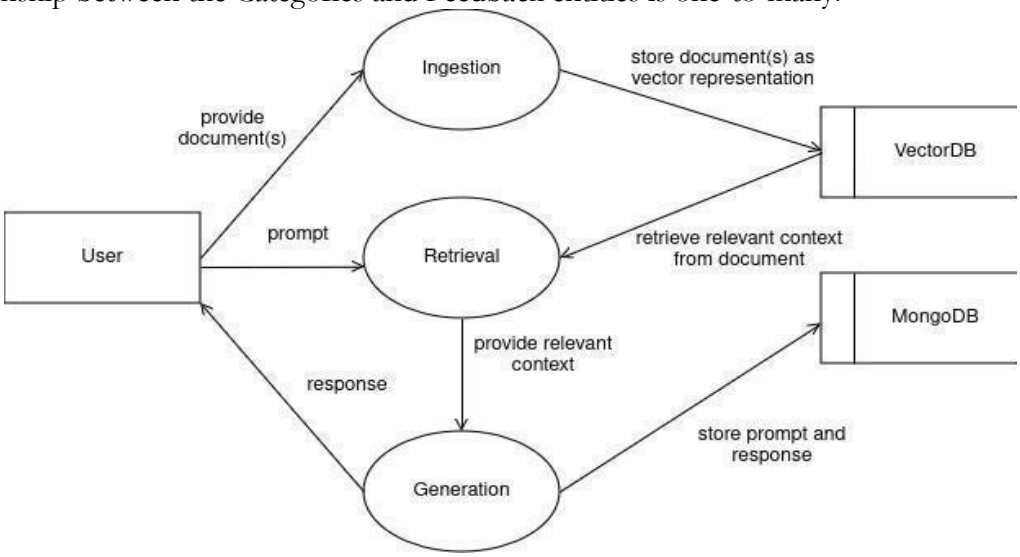


Figure 3: Study flow diagram

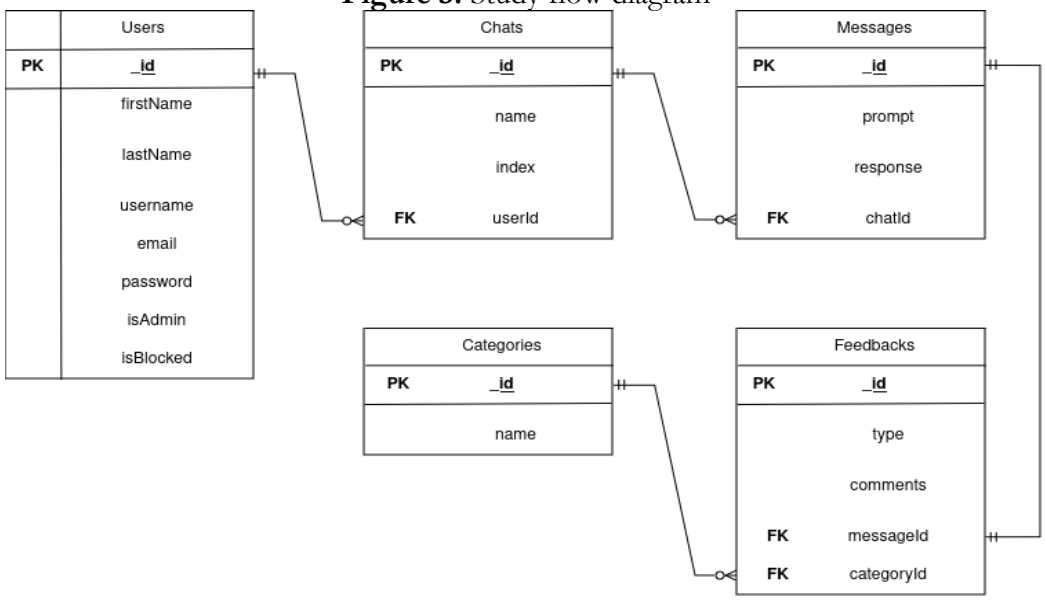


Figure 4: Entity relationship diagram

Development Environment:

Typescript is used as the programming language of choice for this project. The dynamic nature of JavaScript makes it error-prone, causing errors due to vaguely defined variables, which can be avoided by using a type system. Typescript does exactly that, as it enforces strict type safety by defining the types of every variable, it makes sure that no variable is being assigned an incorrect value. Typescript also tightly integrates with VS Code by showing type hints as the code is being written.

The research uses the library Langchain.js (version 0.3.7) for interacting with the LLM and the vector database. The latest versions of NodeJS and Typescript are used, which at the time of writing are 22.12.0, and 5.7.2 respectively. Additionally, Ollama (version 0.3.4) is used for running the Llama 3.2 1B LLM locally.

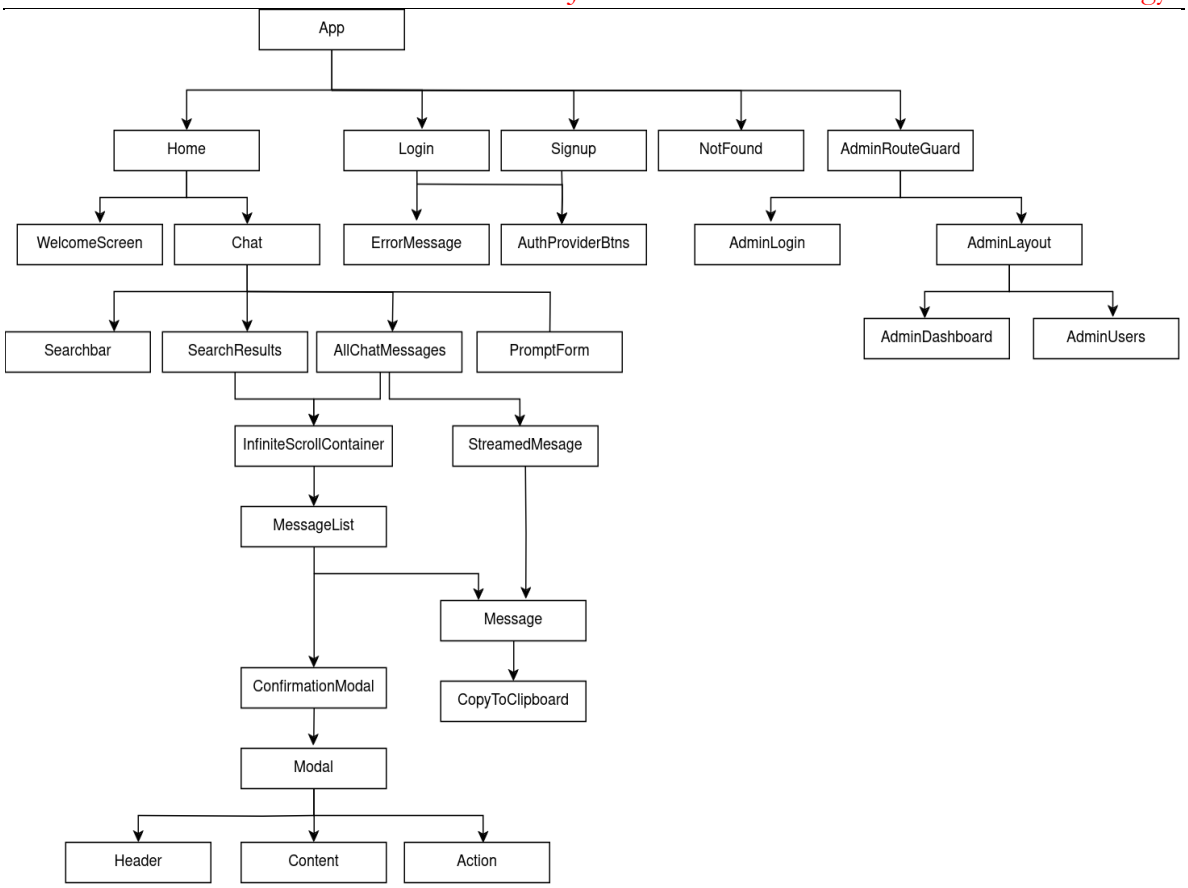


Figure 5: All main components of the proposed system

Integrated Development Environment:

Visual Studio (VS) Code is used as the IDE for the project. VS code is famous for developing web applications because it has a vast community and comes with many extensions which simplify the job of the developer. The customizable nature of VS Code enables developers to customize the IDE according to their tastes.

Version Control:

Git is used to version control the project. Git is an indispensable tool for modern development. The use of Git is widespread in the software development industry. The project's source code is hosted on GitHub.

Repository Setup:

The project is organized into a single directory, with separate directories for the front end and the back end. The source code for both the front end and the back end is included in a single Git repository. Each directory is its npm project, maintaining its dependencies.

During the development of the application for this research, a single main branch was used to push commits for new features. In the future, the addition of new features will follow the pull-request (PR) workflow. Contributors will create a new branch from the main branch and commit their code to this branch; they could request the merging of their feature branch with the main branch by creating a PR on GitHub. The repository owner will review the code, if the code is satisfactory, they will merge it with the main branch, otherwise, they may request additional changes by commenting on specific code snippets.

Modules:

The application is a Single Page Application (SPA), where the client is served a static HTML file about JS. The JS then takes control of the application and is responsible for routing, fetching data, and updating the UI state. The modules of the application are described in terms of the backend and frontend.

Controllers:

The controllers are responsible for handling the business logic for the application. Controllers are defined against HTTP and URLs. The application is composed of the following controllers:

Table 2: Controller types

Controller	Functionalities
Authentication Controller	Responsible for authenticating and authorizing users. It uses session cookies to verify whether a user is logged in or not. It has two controllers, one for login, and another for signup.
Chat controller	Responsible for creating, updating, and deleting chats. It also has a method that enables users to send prompts, the response to which is then streamed to the user using Web sockets
Message controller	Responsible for loading users' chat history. The messages are pagi- noted, and can be sorted by providing the appropriate parameters to the endpoint
Feedback controller	Provides the ability to submit feedback on messages. It also provides a list of paginated feedback which can only be accessed by admin users.
End-user controller	Provides user management functionality, enabling admin user to block/delete other users

Error Handling:

To prevent the server from crashing, the application controllers are wrapped in a global error handler, which responds with a 500-status code if an error occurs. Additionally, any error specific to a controller is handled inside that controller, which responds with the appropriate message and status code.

To ensure data validation, Zod is used, which is a schema validation library. The correct form of the input data for a controller is defined in terms of a Zod schema and only if the schema is matched, the corresponding controller takes an action, otherwise, an error response is sent to the client with an appropriate error message.

Deployment Environment:

The web application will be deployed on a cloud provider. For the current system, Google Cloud Platform (GCP) was selected as the cloud platform. GCP provides numerous cloud offerings, spanning from IaaS to PaaS. For this application, the IaaS offering, Cloud Compute Engine will be used.

The application server uses Ubuntu as the Operating System (OS) and has Nginx installed. The Nginx server points to the running Express.js application, which is being executed in the NodeJS runtime. The application server can communicate with the database server using the TCP/IP protocol suite. The UI of the application is hosted on a Netlify server, the Netlify server uses a managed OS since they do not disclose which OS is being used, inside the OS, the React application is in the form of three files, index.html, styles.css, and script.js. The CSS styles and JS files are bundled and minified during the build process. The user web browser fetches the index.html, which requests for the styles, and script.js files from the Netlify server. The script.js takes control of the application after this process and interacts with the application server. The deployment diagram for the application is illustrated in Figure 6.

The Nginx server acts as a reverse proxy for the Express.js application and enhances security by hiding the port on which the Express.js application is running. Nginx runs by default on port HTTP (port 80) and HTTPS (port 443), allowing the front end to send requests directly using the API URL without specifying a custom port. Furthermore, to scale the application

horizontally, in the future, it could be hosted on multiple servers in multiple regions to distribute the load amongst these servers, Nginx can act as a load balancer in this scenario by routing traffic to an appropriate server based on the network load.

Testing & Results:

Frontend: The testing was carried out in three stages. In the first stage, unit tests were written for the smallest components in the component tree (leaf nodes). In the second stage, integration tests were written to make sure the components were interacting with each other correctly. In the last stage, end-to-end testing was performed to test that the whole front end was operating correctly. The component tree for the application is illustrated in Figure 5.

Tools: Vitest was used as the test runner, with react-testing-library being used to test the workings of the React components. JSDOM was used to emulate the browser DOM in the Node.js runtime. For end-to-end tests, Cypress was used.

Table 3: Testing

Unit Testing	Integration Testing	End-to-End Testing
Unit tests were written for the components that are at the leaves of the component tree shown in Figure 5. The leaves of the component tree are at the lowest level of the tree and are components that do not utilize other components. The test results are summarized in Figure 7.	Integration tests were written for the non-leaf components of the component tree. These components are utilizing the components below it. Therefore, it was imperative to test that the lower-level components were integrating correctly with the upper-level components. The test results are summarized in Figure 8.	During the end-to-end testing, using a browser automation tool, the application was thoroughly checked as if an actual user was using the application. The test results are summarized in Figure 9.

```
> vitest tests/unit/

 DEV  v1.6.0 /home/asadkhan/Downloads/Computer_Science_UOP/Seventh_Semester/FYP-I/FYP/client

 ✓ tests/unit/InfiniteScrollContainer.test.tsx (5)
 ✓ tests/unit/useIntersectionObserver.test.tsx (5)
 ✓ tests/unit/PaginatedTable.test.tsx (4) 605ms
 ✓ tests/unit/PromptForm.test.tsx (4) 335ms
 ✓ tests/unit/CopyToClipboardButton.test.tsx (5)
 ✓ tests/unit/Searchbar.test.tsx (5) 350ms
 ✓ tests/unit/PositiveFeedbackForm.test.tsx (4) 388ms
 ✓ tests/unit/EditChatForm.test.tsx (5) 370ms
 ✓ tests/unit/AuthProviderBtns.test.tsx (3)
 ✓ tests/unit/NegativeFeedbackForm.test.tsx (4) 421ms
 ✓ tests/unit/NegativeFeedbackButton.test.tsx (2)
 ✓ tests/unit/PositiveFeedbackButton.test.tsx (2)
 ✓ tests/unit/ErrorMessage.test.tsx (1)
 ✓ tests/unit/WelcomeScreen.test.tsx (1)
 ✓ tests/unit/LoadingIcon.test.tsx (1)

 Test Files  15 passed (15)
      Tests  51 passed (51)
   Start at  19:04:41
   Duration  4.06s (transform 763ms, setup 1.84s, collect 4.12s, tests 3.06s, environment 7.63s, prepare 2.05s)

 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

Figure 7: Frontend Unit Testing Results

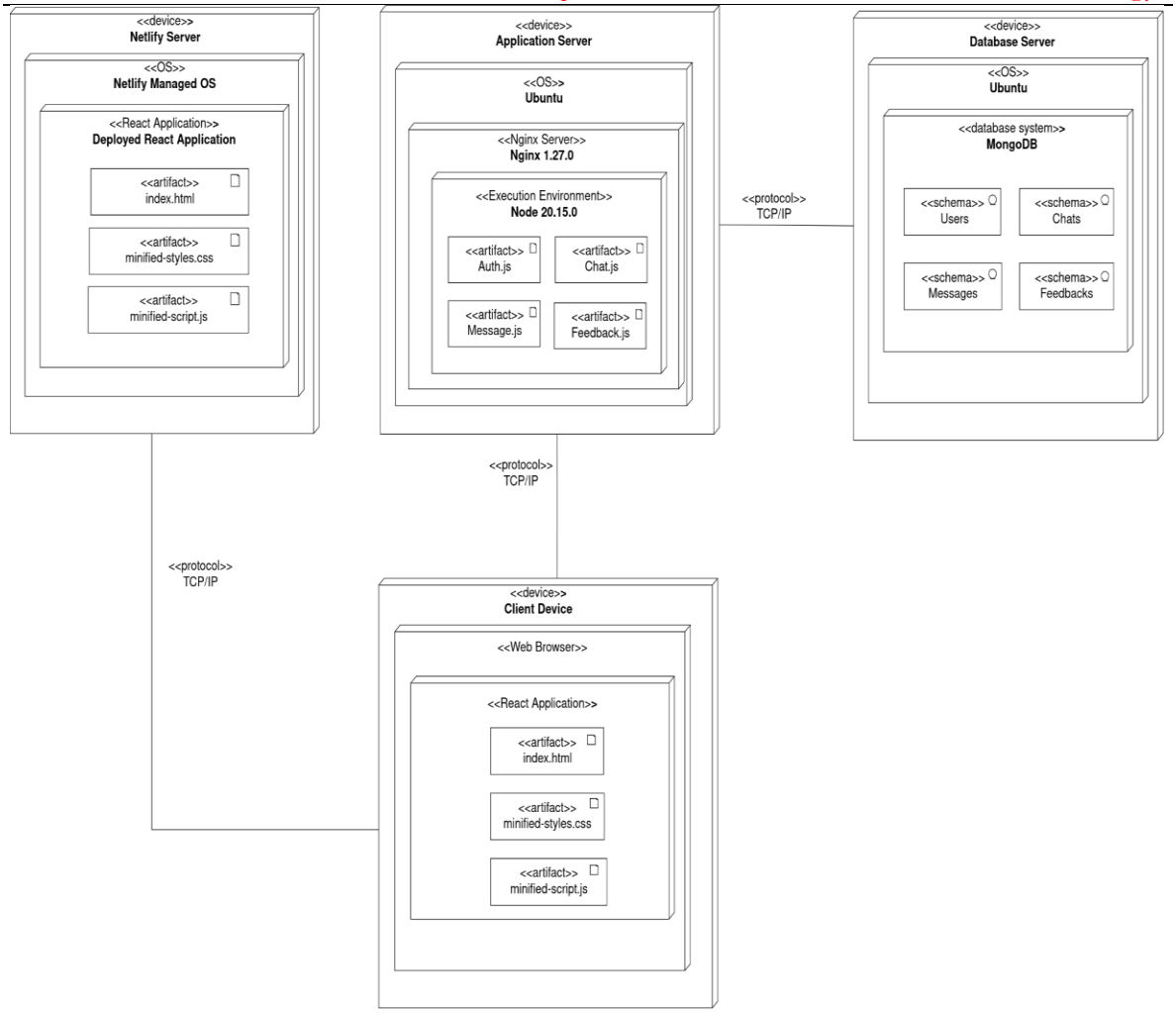


Figure 6: Deployment diagram

```
✓ tests/integration/Signup.test.tsx (13) 2669ms
✓ tests/integration/Login.test.tsx (8) 1051ms
✓ tests/integration/SearchResults.test.tsx (4) 384ms
✓ tests/integration/AllChatMessages.test.tsx (4)
✓ tests/integration/AdminUsers.test.tsx (3) 1202ms
✓ tests/integration/Sidebar.test.tsx (4) 574ms
✓ tests/integration/CreateChatModal.test.tsx (7) 387ms
✓ tests/integration/ChatLink.test.tsx (6) 431ms
✓ tests/integration/Message.test.tsx (8)
✓ tests/integration/Home.test.tsx (4) 307ms
✓ tests/integration/StreamedMessage.test.tsx (4)
✓ tests/integration/NegativeFeedbackModal.test.tsx (4) 391ms
✓ tests/integration/MessageList.test.tsx (2)
✓ tests/integration/Chat.test.tsx (3) 356ms
✓ tests/integration/PositiveFeedbackModal.test.tsx (4)
✓ tests/integration/AdminDashboard.test.tsx (1) 305ms
✓ tests/integration/CreateChatButton.test.tsx (3)
✓ tests/integration/ConfirmationModal.test.tsx (1)

Test Files  18 passed (18)
     Tests  83 passed (83)
  Start at  19:04:52
  Duration  6.44s (transform 880ms, setup 1.90s, collect 8.39s, tests 9.30s, environment 10.12s, prepare 2.87s)

PASS  Waiting for file changes...
      press h to show help, press q to quit
```

Figure 8: Frontend Integration Testing Results

```
Tests:        5
Passing:      5
Failing:      0
Pending:      0
Skipped:      0
Screenshots:  0
Video:        false
Duration:     5 seconds
Spec Ran:     signup.cy.ts
```

```
(Run Finished)
```

Spec		Tests	Passing	Failing	Pending	Skipped
✔ home.cy.ts	00:13	16	16	-	-	-
✔ login.cy.ts	00:04	6	6	-	-	-
✔ signup.cy.ts	00:05	5	5	-	-	-
✔ All specs passed!	00:23	27	27	-	-	-

Figure 9: End-to-End Testing

Backend: The backend was tested using mostly end-to-end tests, making sure each API endpoint was working as intended. The details of the testing are described below.

Tools: Super test was used as the HTTP client to interact with the API endpoints, with Jest being used as the test runner. Instead of using an actual database, an in-memory alternative provided by the package MongoDB-memory-server was used.

Strategy: Each API endpoint was tested by making sure they only allow authenticated users to access them. The API endpoints which required user data were tested to make sure the data was being validated correctly, and if the user provided incorrect data (wrong type or incomplete data) the API responded with the correct status code and a feedback message.

The API endpoints which are only meant to be accessed by the admins were also tested to make sure only authorized users (admins) were able to access them, and other users received a 401 (Unauthorized) status code when they tried to access these endpoints.

End-to-End Testing: Each API endpoint was tested in detail, verifying the correct behavior of the endpoint, and making sure it only accepted valid input. The access roles for each endpoint were also tested, to make sure only authenticated and authorized users could access specific routes. The test results are summarized in Figure 10.

```
(node:42047) ExperimentalWarning: VM Modules is an experimental feature and might change at any time
(Use `node-20 --trace-warnings ...` to show where the warning was created)
PASS tests/login.test.ts (6.236 s)
(node:42035) ExperimentalWarning: VM Modules is an experimental feature and might change at any time
(Use `node-20 --trace-warnings ...` to show where the warning was created)
PASS tests/signup.test.ts (6.333 s)
PASS tests/feedback.test.ts (6.719 s)
PASS tests/message.test.ts (6.689 s)
PASS tests/user.test.ts (6.873 s)
PASS tests/chat.test.ts (7.048 s)
A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper t
n. Try running with --detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.

Test Suites: 6 passed, 6 total
Tests:       71 passed, 71 total
Snapshots:   0 total
Time:        7.829 s, estimated 12 s
Ran all test suites related to changed files.
```

Figure 10: End-to-End Backend Testing Results

There are many limitations in existing LLMs without a RAG sub-system, our proposed large language models with RAG try to overcome these limitations. Comparison between large language model without RAG and with RAG is discussed below in Table 4

Table 4: Comparison of LLMs without RAG and LLMs with RAG

LLM without RAG	LLM with RAG
Lack of contextual information	The LLM response is based on user-provided context (their documents)
Unable to accommodate newer information	Users can provide new information to the LLM as documents.
Prone to hallucination	The tendency of LLM to hallucinate information that was not included in the model's training data is curbed by augmenting LLM's response with user-provided document(s)
Limited by knowledge cut-off date	Users can provide newer information to the LLM by uploading it as a document.
Requires re-training to learn new information The model is limited to publicly available data	Do not have to re-train the model, instead, the user prompt is augmented with the new information. Users can interact with the LLM in the context of their private data.

Discussion:

The improvements to the Retrieval Augmented Generation (RAG) workflow, organized based on the stages of the workflow, are presented in the sections below.

Indexing:

The objective of indexing is to convert the user document into a concise representation that can then be used to retrieve relevant context during retrieval. Following improvements could be made to further facilitate the effective retrieval of context.

Chunking:

The project uses recursive chunking to make sure sentences are not split in the middle, causing a loss of context. While this methodology is effective, it has its drawbacks, for instance, the retrieval of text chunks depends upon their size, so if you have longer text chunks then you have more context, but the embedding vectors for these chunks are more prone to be matched for any user query. There are other far more efficacious chunking methods, such as small-to-big and sliding windows [16]. The small-to-big utilizes the best of both worlds, by matching the user query with small text chunks and then returning the longer context that contains the matched text chunk. Thus enabling effective retrieval and availability of better context [16].

Metadata:

Meta-data such as the document title, or keywords in the text chunk can be stored with the text chunks obtained from user document(s), to further facilitate accurate retrieval[16].

Structured Data:

The current system works best for unstructured data, particularly textual data, it is ill-suited for structured data, especially in the format of tables in documents, or SQL databases. This limitation can be overcome by integrating table querying into the retrieval step of the system. It contains a set of documents, along with SQL tables containing additional relevant information. A router is then integrated into the system, as discussed in Section 6.3.2, to determine during the retrieval process whether a user query can be satisfied from information contained in an SQL table. If the router selects the SQL table, then the user query is passed to an LLM to create an SQL query, which queries the table and returns the result, which is then passed to the generator to produce the final response [17].

Retrieval:

The main goal of the retrieval step is to provide the generator with the appropriate context. This step can be further improved as follows.

Query Rewriting:

Original queries often suffer from underperformance because of poor expression or lack of semantic information [16]. By integrating query rewriting into the RAG workflow, the

performance can be improved further. Query rewriting works by asking a Large Language Model (LLM) to rewrite the original query to be more expressive, before eventually passing the revised query to the retrieval sub-system [16].

Query Decomposition:

A weakness of the current system is that it retrieves semantically similar text chunks based on a single query. This approach is insufficient to answer queries meant to be answered from different portions of the corpus. A composite query is decomposed into sub-queries, which are then passed to the retrieval sub-system to retrieve appropriate text chunks from the index [16]. An LLM then generates answers for each sub-query, which are then passed again to the LLM as the context for the composite query during the final generation stage [17].

Query-to-Document Expansion:

The difference in grammar between the user query and the text chunks in the vector index can result in incorrect context being retrieved from the vector index. To alleviate this problem, query-to-document expansion is proposed, in this approach, the user query is passed as is to the LLM which generates a hypothetical answer to the query, this response is then converted into a vector representation and its similarity with the vector index is checked, to retrieve the relevant context. The idea behind this technique is that the LLM will generate a hypothetical/hallucinated response, which would be closer in format to the relevant context in the vector index [17]. This process is known as Generation-Augmented Retrieval (GAR) [18].

Routing:

One or more of these advanced retrieval methods might be required, depending on the user query. To help the system, decide which technique(s) should be used, a routing subsystem could be incorporated into the system. The routing subsystem passes the user query to an LLM along with a strict template prompt, which instructs the model to choose between the different technique(s) to be used based on the query. The response is then used to decide which technique(s) should be applied to the user query [17].

Re-Ranking:

LLMs are prone to get lost in the sea of information passed to them in their context. This is known as the lost-the-middle problem.[10] The solution to this problem is to include the most relevant text chunks at the beginning or end of the context. To achieve this, we could use two embedding models during the retrieval process. In the initial stage, retrieve relevant text chunks based on a smaller embedding model, thus retrieving k text chunks. These text chunks, along with the user query, are then embedded with a larger, more semantically rich embedding model, and the list of text chunks is re-ranked based on its similarity with the user query. This results in the most relevant text chunks being at the top of the list [17].

Generation:

The primary objective of the generator sub-system is to generate a response given a user query and some context. Further improvements to this step are discussed in the sections below.

Context Consolidation:

The system uses the simplest way of just concatenating together the retrieved text chunks. However, this approach has two problems: first, the text chunks combined might be greater than the LLM's context window size, second, the information may not be optimally located, the concatenated text chunks might be disparate and disconnected [17].

Using context consolidation, which better synthesizes the retrieved text chunks to overcome the aforementioned problems. Some common techniques include: passing each text chunk to an LLM to summarize each, and additionally creating a global summary of all the retrieved text chunks. This greatly reduces the length of the input context and forms the text chunks into a coherent and self-consistent context [17].

Multi-modality:

The current system uses a textual LLM, which is only effective for textual documents. However, documents often contain Figures, such as diagrams, charts, and images. The existing system cannot accommodate RAG on non-textual modalities such as audio, video, and images. There have been recent advancements in multimodal LLMs that have enabled these models to handle information in multiple formats, thus allowing the development of general-purpose generators [19]. To implement multi-modal RAG, a multi-modal index should be created. This requires the usage of a multi-modal embedding model, which can embed both textual and non-textual information into a vector space. However, the current multimodal embedding models are only capable of representing text and images in a unified space. Mapping multi-modal information into a unified space remains a challenge [19].

Practical Implications:

This research addresses the need to enhance user interactions with their documents by integrating large language models (LLMs). LLMs are provided with user document context thus enabling users to gain insights from their conversations with the system. Users can interact with their documents using natural language. The language model can answer questions from the user documents' context, rather than from other sources which might not be relevant to the user. Users are then able to efficiently find the information they are looking for instead of having to explore the document themselves.

Users are provided with a rich experience while interacting with written material. Instead of having to reread document(s), users can ask the LLM to summarize, generate questions, or perform other generative tasks based on their document(s).

Conclusion:

The existing application's drawback is that PDF files containing pictures are not able to be shown in the dry run section. To deal with this issue, we plan to explore various algorithmic techniques to manipulate and exhibit picture-rich PDFs in the dry run segment. In addition, we intend to offer a download option that would permit users to save a replica of their questions on their tendency. Furthermore, future developments may include a group of users from various specialties to conduct tests and assess the accuracy and accomplishment of the Chatbot system.

Conflicts of Interest:

There are no conflicts of interest, according to the authors.

References:

[1] and R. S. F. Farhi, R. Jeljeli, I. Aburezeq, F. F. Dweikat, S. A. Al-shami, "Analyzing the students' views, concerns, and perceived ethics about chat GPT usage," *F. Farhi, R. Jeljeli, I. Aburezeq, F. F. Dweikat, S. A. Al-shami, R. Slamene*, vol. 5, p. 100180, 2023, doi: https://doi.org/10.1016/j.caeai.2023.100180.

[2] I. Adeshola and A. P. Adepoju, "The opportunities and challenges of ChatGPT in education," *Interact. Learn. Environ.*, Sep. 2023, doi: 10.1080/10494820.2023.2253858.

[3] M. B. & F. G. Olaf Zawacki-Richter, Victoria I. Marín, "Systematic review of research on artificial intelligence applications in higher education – where are the educators?," *Int. J. Educ. Technol. High. Educ.*, vol. 16, p. 39, 2019, doi: https://doi.org/10.1186/s41239-019-0171-0.

[4] T. Li, E. Reigh, P. He, and E. Adah Miller, "Can we and should we use artificial intelligence for formative assessment in science?," *J. Res. Sci. Teach.*, vol. 60, no. 6, pp. 1385–1389, Aug. 2023, doi: 10.1002/TEA.21867.

[5] and B. Y. P. G. Verma, T. Campbell, W. Melville, "Navigating Opportunities and Challenges of Artificial Intelligence: ChatGPT and Generative Models in Science Teacher Education," *J. Sci. Teach. Educ*, vol. 34, no. 8, pp. 793–798, 2023, doi: https://doi.org/10.1080/1046560X.2023.2263251.

[6] M. Al-Emran, "Unleashing the role of ChatGPT in Metaverse learning environments: opportunities, challenges, and future research agendas," *Interact. Learn. Env.*, pp. 1–10,

2024, doi: https://doi.org/10.1080/10494820.2024.2324326.

[7] Z. H. & Y. G. Yuren Mao, Yuhang Ge, Yijiang Fan, Wenyi Xu, Yu Mi, "A survey on LoRA of large language models," *Front. Comput. Sci*, vol. 19, pp. 1–140, 2024, doi: https://link.springer.com/article/10.1007/s11704-024-40663-9.

[8] and H. W. Gao, Yunfan Xiong, Yun Gao, Xinyu Jia, Kangxiang Pan, Jinliu Bi, Yuxi Dai, Yi Sun, Jiawei Meng Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," *arXiv:2312.10997*, 2023, [Online]. Available: https://arxiv.org/pdf/2312.10997

[9] P. P. Ray, "ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," *Internet Things Cyber-Physical Syst.*, vol. 3, pp. 121–154, 2023, doi: https://doi.org/10.1016/j.iotcps.2023.04.003.

[10] J. Zhou, P. Ke, X. Qiu, M. Huang, and J. Zhang, "ChatGPT: potential, prospects, and limitations," *Front. Inf. Technol. Electron. Eng.*, vol. 25, no. 1, pp. 6–11, Jan. 2024, doi: 10.1631/FITEE.2300089/METRICS.

[11] and S. R. Z. Ahmad, W. Kaiser, "Hallucinations in ChatGPT: An Unreliable Tool for Learning," *Rupkatha J. Interdiscip. Stud. Humanit*, vol. 15, no. 4, 2023, doi: https://doi.org/10.21659/rupkatha.v15n4.17.

[12] J. M. P.-A. and A. Labisa-Palmeira, "Quick Review of Pedagogical Experiences Using Gpt-3 in Education," *J. Technol. Sci. Educ*, vol. 14, no. 2, pp. 633–647, 2024, doi: DOI: https://doi.org/10.3926/jotse.2111.

[13] and L. C. Z. Xie, X. Evangelopoulos, Ö. H. Omar, A. Troisi, A. I. Cooper, "Fine-tuning GPT-3 for machine learning electronic and functional properties of organic molecules," *Chem. Sci*, vol. 15, no. 2, pp. 500–510, 2023, [Online]. Available: https://pubs.rsc.org/en/content/articlelanding/2024/sc/d3sc04610a

[14] M. Enis, "From LLM to NMT," 2024, doi: 10.36934/TR2024_197.

[15] and G. T. L. Caruccio, S. Cirillo, G. Polese, G. Solimando, S. Sundaramurthy, "Claude 2.0 large language model: Tackling a real-world classification problem with a new iterative prompt engineering approach," *Intell. Syst. with Appl.*, vol. 21, p. 200336, 2024, doi: https://doi.org/10.1016/j.iswa.2024.200336.

[16] X. H. Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, Ruicheng Yin, Changze Lv, Xiaoqing Zheng, "Searching for Best Practices in Retrieval-Augmented Generation," *arXiv:2407.01219*, 2024, doi: https://doi.org/10.48550/arXiv.2407.01219 Focus to learn more.

[17] E. Kasneci *et al.*, "ChatGPT for good? On opportunities and challenges of large language models for education," *Learn. Individ. Differ.*, vol. 103, p. 102274, Apr. 2023, doi: 10.1016/J.LINDIF.2023.102274.

[18] W. C. Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, "Generation-augmented retrieval for open-domain question answering," *arXiv:2009.08553*, 2021, doi: https://doi.org/10.48550/arXiv.2009.08553.

[19] S. J. Ruochen Zhao, Hailin Chen, Weishi Wang, Fangkai Jiao, Xuan Long Do, Chengwei Qin, Bosheng Ding, Xiaobao Guo, Minzhi Li, Xingxuan Li, "Retrieving Multimodal Information for Augmented Generation: A Survey," *Find. Assoc. Comput*, pp. 4736–4756, 2023, doi: 10.18653/v1/2023.findings-emnlp.314.



Copyright © by Talha Ahsan et al and 50Sea. This work is licensed under Creative Commons Attribution 4.0 International License.