**AI-Enhanced Database Management System**

Software & Thesis submitted to Swansea University in Partial Fulfilment for the Degree of Master of Science

Department of Computer Science’s, Swansea University

**Mutasim El-Khidir (2254626)**

**Submission date: September 2024**

**Summary**

The internet is a vast collection of databases, with browsers serving as user interfaces to access and navigate this information. While browsing has become intuitive, interacting with specialized databases, such as those in healthcare, remains challenging and inefficient. Many databases require user training, are prone to errors, and often feel unintuitive due to poor design, creating significant barriers to efficient use.

Humans have evolved to express and exchange ideas through natural language, which remains the primary form of knowledge exchange—as evidenced by this dissertation alongside the program created for this project.

Building on this natural method of communication, the project leverages NLP to simplify database interaction through a user-friendly search interface, capitalizing on users’ familiarity with voice assistants, chatbots, and tools like ChatGPT. By allowing users to type or dictate queries in their natural language, the system translates them into commands that retrieve, compare, validate, and manipulate data across multiple databases. This eliminates the need for direct interaction with complex databases, reducing the likelihood of errors and the cognitive load on users.

Designed with modularity, the system allows for the easy integration of additional features. Open-source tools and third-party APIs underpin its functionality, enabling advanced capabilities without extensive custom development. These components handle various data formats, support voice commands, and streamline data processing, making the solution adaptable to diverse environments.

This approach demonstrates the power of NLP to improve database usability, particularly in critical sectors like healthcare, where efficient data management can have life-saving impacts.

**Declarations & Statements**

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

This dissertation is the result of my own independent work, except where otherwise stated. Other sources are acknowledged by giving explicit references.

I hereby give consent for my dissertation, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

**Mutasim El-Khidir**

**XX/09/2024**

**Table of Contents**

**Table Item:**  **Page:**

1. Summary 2
2. Declarations & Statements 3
3. Table of Contents 4
4. Figures & Tables 5
5. Abbreviations 6
6. Introduction
7. Background
8. Literature review
9. Software: AI Assistant
10. Evaluation
11. Discussion
12. Conclusion
13. References
14. Appendices

**Figures & Tables**

**Figures: Page:**

1. Figure 1: Graphic User Interface (GUI)
2. Figure 2: Currently used GUI
3. Figure 3: Correlation Analysis Results Page 25
4. Figure 4: Process/Data Flow
5. Figure 5.1: Program Class UML: main.py
6. Figure 5.2: Program Class UML: file\_handler.py
7. Figure 5.3: Program Class UML: comparison.py
8. Figure 5.4: Program Class UML: query\_handler.py
9. Figure 5.5: Program Class UML: voice\_assistant.py
10. Figure 5.6: Program Class UML: gui\_handler.py
11. Figure 5.7: Program Class UML Diagram

**Tables:**

1. Table 1: GPT Model comparison

**Abbreviations**

**Technology and Programming:**

1. **AI:** Artificial Intelligence
2. **CRC:** Class-Responsibility-Cards
3. **NLP:** Natural Language Processing
4. **LLM:** Large Language Model
5. **GUI:** Graphical User Interface
6. **HCI:** Human-Computer Interaction
7. **SQL:** Structured Query Language
8. **DBMS:** Database Management System
9. **CSV:** Comma-Separated Values
10. **DOCX:** Microsoft Word Open XML Document
11. **PDF:** Portable Document Format
12. **UML:** Unified Modelling Language

**Cloud and Infrastructure**

1. **AWS:** Amazon Web Services
2. **API:** Application Programming Interface (or Application Program Interface)

**Other:**

1. **NHS:** National Health Service

**Introduction**

Evidence shows that employees waste considerable time operating database systems for administrative and operational purposes. This avoidable inefficiency has a negative rippling effect on all industries, with very few exceptions. (Admin Matters: The Impact of NHS Administration on Patient Care, n.d.)

In a labour—and capital-intensive industry like healthcare, operational efficiency saves lives before money. Reducing administrative costs (time) means serving more patients without hiring more staff, increasing staff satisfaction, and saving money.

For example, doctors are not trained to use database systems as part of their educational curriculum. Even if software usage is included in the academic material, a new release or update may require new learning material and training. A newly hired doctor might want to extract patient information during a meeting in a hypothetical scenario. However, if the doctor cannot effortlessly/intuitively use the available database systems, the doctor and patient will suffer.

Well-designed database systems are intuitive to use. However, creating an intuitively navigable database and information system that meets its inception purpose requires ingenuity, time, money, and all sorts of resources that are not readily available, especially in an industry like healthcare (NHS). This is evidenced by the data losses reported during the COVID-19 pandemic. (Kelion, 2020)

Designing an intuitive user interface requires a deep understanding of the end user. This is not easily achieved when factoring in user perception to visual representation, pattern recognition, preferences, and other elements affecting the user experience while using a database.

Siri, Google Assistant, and Alexa create an experience that feels intuitive, natural, and effective, reducing the cognitive load on users. A similar setup may serve as a fallback solution for an uneducated user. Capitalising of the widespread knowledge in interacting with these assistants, including Chatbots.

**Background**

AI solutions in information systems and database management have been utilised on various levels for decades. Before the invention of Tone Dialling, human operators connected the phone lines manually after asking the user whom they wanted to speak with. Arguably, tone-dial technology was the foundation of sound recognition, and a telephone switchboard is a database management system, with the operator acting as a Database Administrator.

1952, Bell Labs created the first speech-recognition device (Audrey) to recognise spoken digits. Currently, one example of advancements in NLP is the commercialisation of AI voice assistants, which can understand many accented languages and perform complex tasks on behalf of the user.

Contemporary commercial application examples include chatbots, telephone directories (voice-activated), and other AI-collected and processed information, i.e., social media listeners. These applications have eliminated the need for a human operator to perform a highly predictable and automatable task, saving money and raising efficiency by reducing administrative workloads.

The same NLP technology used to generate profit for technology giants like Apple, Google, and Amazon can eliminate tedious and automatable tasks in most workplaces. By integrating NLP to translate automatically and action the user's query/task in natural language, workloads are reduced and eliminated.

Emerging AI solutions for database management harnessing powerful AI tools are commercialised to address the needs of a growing market with massive amounts of data that may benefit from a novel approach. Examples of commercialised products that perform a similar task are Amazon Aurora, Oracle Digital Assistant, Microsoft Azure SQL Database, and others.

However, the administrative time waste issue remains prevalent, whilst the available technology does not address the needs of a population that may seem unaware of the potential AI and NLP can bring to the workplace with HCI in mind.

Professor Yannis Ioannidis said, “The database and HCI communities live on their separate islands with essentially no ferry line between them.” (Ioannidis, 1996) Academic material addressing this specific angle took much work to find. Nevertheless, searching by keyword has surfaced interesting material dating back to when the technology had not matured or reached today's levels by comparison. With much of the research being completed and Intellectually Protected by the tech giants for profitability, the released technologies will have addressed many aspects of this research. However, the scientific material and software remain proprietary.

The primary search tool used to find the literature is Google Scholar, which generated results from reputable academic sources. However, the research also considered tech companies and potential users. The keywords used in the literature search process have multiple variations and combinations, but the core terms were NLP, AI, DBMS, and HCI.

Finding scientific literature on the subject was challenging because companies hold proprietary rights over scientific research and the tools needed to achieve the project goals. While the proposal may be reinventing the wheel compared to the number of products advertised performing virtually similar functions. This research should provide a simplified version of the “mechanics” and underlying technology behind these products.

However, the literature found was enlightening. The possible applications and areas of AI that can enhance databases seem infinite. Meanwhile, the DBMS HCI element is widely ignored outside of commercial space. An example would be harnessing the power of Google Voice Assistant, Siri, or Alexa in a working environment. A hypothetical example would be a doctor pulling patient information through a voice command in natural language, then having all the information the doctor needs at that time presented to the doctor. The impact of an efficient system saves lives in an industry like healthcare. (Healthcare UX: When UX Hurts And Even Kills)

The research papers examined demonstrate the progress made in the relevant fields of the ongoing research, while highlighting the need and potential for utilising these technologies in various scenarios to raise efficiency in most industries. While healthcare has been repeatedly mentioned and researched in this paper, it is the most appropriate example, while actual use cases extend beyond healthcare.

**Literature Review**

Finding scientific literature on the subject was challenging because companies hold proprietary rights over scientific research and the tools needed to achieve the project goals. While the proposal may be reinventing the wheel compared to the number of products performing virtually similar functions, going into detail should prove otherwise on many levels or show the unique selling point of the project in hand.

However, the literature found was enlightening. The possible applications and areas of AI that can enhance databases seem infinite. Meanwhile, the DBMS HCI element is widely ignored outside of commercial space. An example would be harnessing the power of Google Voice Assistant, Siri, or Alexa in a working environment. A hypothetical example would be a doctor making the rounds and speaking with patients, does not have to go back to log the interactions nor the observations as they are processed in the background in real-time, with less room for human error.

The research papers examined demonstrate the progress made in the relevant fields of the ongoing research while highlighting the need and potential for utilising these technologies in various scenarios to raise efficiency in most industries. While healthcare has been repeatedly mentioned and researched in this paper, it is the most appropriate example, while actual use cases extend beyond healthcare.

**Software: AI Assistant**

Initially imagined as a search bar with space to type or dictate a query in natural language, the following GUI was the expected outcome and product:

A rectangular black and white rectangular object

Description automatically generated

**Figure 1**

The above interface seems insufficient for generating text responses or to have to option to read the answer instead of hearing it. Unless it opens other windows and has more interactive GUI functions that are time-consuming to develop and test, considering the research time frame. Thus, the below currently used GUI was selected:

A screenshot of a computer

Description automatically generated

**Figure 2**

While visually less attractive, the functionalities are easier to test in the above format, considering the ongoing testing and development of the program from the initial design and the time needed to utilise sophisticated GUI tools.

The below diagram acted as a simplistic process/data flow of the search bar:



**Figure 3**

Many visualisations were created in the development of the program, with various levels of abstraction and detail. Some were created with Software Engineering principles in mind, as in the UML and CRC diagrams and visualisations, while other visualisations serve as a guide and provide general explanations.

The graph below should express the general logic/functions and procedures of the current program iteration in a simplistic format:

A diagram of application software

Description automatically generated

**Figure 4**

The current program iteration is a GUI based application developed using the Tkinter framework, designed to facilitate file handling, data comparison, query processing, and voice-assisted interaction. At its core, the application allows users to load multiple data files, compare their contents, query the data, and receive responses through text or voice. The modular structure of the application ensures ease of use, flexibility, and expandability, making it a versatile tool for data-centric tasks.

While the program is being developed as part of research from scratch, there were no set classes or plans aside from Figures 1 & 3 and theory; the classes and methods are poorly designed and have room for improvement. The research nature of the project creates a challenge in designing the classes and class interactions. While the project is still undergoing development, the design will gradually align with the generally acceptable practices and principles of software engineering as the project matures with future iterations.

**Software Engineering, Design & Architecture**

The current set-up is composed of Six classes + an environment file. The breakdown is:

A screenshot of a computer program

Description automatically generated

**Figure: 5.1**

**main.py** (corresponding to the Application in Figure 4) defines and controls the core logic for the application in terms of:

Loading and managing multiple data files in various formats.

Comparing the loaded data frames (tabular & text data structures)

Processing user queries related to the loaded data

Optionally provides voice assistant functionality for hands-free interaction

Integrates with a graphical user interface (GUI) for user interaction.

Overall, this class provides the foundation for the application that can be controlled through a user interface and optionally supports voice interaction.

The **file\_handler.py** class handles the loading of data from various file types into DataFrames, managing databases through SQL, and generating SQL commands using LangChain to handle file-based and database interactions efficiently.

The key components are:

Initialisation: Takes an optional file\_path or connection\_string, creates a database engine, and loads a file into a Pandas DataFrame.

File Loading: Supports Excel, CSV, PDF, DOCX, and SQL files, converting them to DataFrames using relevant libraries like Pandas, pdfplumber, and docx.

SQL Interaction:

Creates a SQLAlchemy engine for SQL files or connection strings. Loads SQL files or queries and executes them, returning results as DataFrames.

Language Model: Uses ChatOpenAI to generate SQL based on natural language prompts and execute database management tasks, using a low setting in the Temperature of the language model to limit “hallucinations”.

A screenshot of a computer program

Description automatically generated

**Figure: 5.2**

A screenshot of a computer

Description automatically generated

**Figure: 5.3**

The **comparison.py** classcompares multiple DataFrames and summarises their differences. With a protocol to handle a single file upload. Key components include:

**Text Comparison:** If the DataFrames contain text content, it checks for differences between them.

**Tabular Comparison**: Converts DataFrames to CSV format and compares their tabular content.

DataComparer efficiently compares text and tabular data, summarising differences across DataFrames.

A screenshot of a computer

Description automatically generated

**Figure: 5.4**

The **query\_handler.py** class handles natural language queries using the ChatOpenAI model and maintains conversation history through the following:

Sets up the ChatOpenAI model at a medium Temperature to accommodate for questions beyond the literal meaning and tracks questions and answers in a history list.

(ask\_question): Processes a question based on DataFrame or string content, generates a response, updates the history, and returns the conversation.

Prompt Creation: Builds prompts using the content (DataFrame or summary) and conversation history.

Response Generation: Sends the prompt to the language model to get an answer.

History Management: Updates the history and retrieves the most recent conversation.

The **voice\_assistant.py** class enables voice-based interactions by combining speech recognition, text-to-speech, and AI language processing. Serving the program by:

Setting up the language model (at a high Temperature to allow for creative responses) and a text-to-speech engine.

Initialises a microphone for voice input and captures and converts voice input to text using Google Speech Recognition.

Listens, standardises, and returns user voice input as text. Refines the captured text using an AI language model.

Converts text to speech and responds audibly. Speaks the response back to the user.

Halts ongoing speech output.

In short, it listens to user input, processes it, and responds with a synthesised speech-language model.

A screenshot of a computer

Description automatically generated

**Figure: 5.5**

A screenshot of a computer

Description automatically generated

The **gui\_handler.py** class manages the graphical user interface for an AI assistant application using Tkinter:

Initialisation: Sets up the main window, including a dark theme, and creates essential GUI widgets. Handles window closing to stop any ongoing operations.

Creates buttons with tooltips for additional information. Sets up the query input area with buttons for file operations and voice input.

Adding further functionalities to the program to Avoid duplicate submissions, Populate the query entry, Change the button's appearance, Update the UI with the response, and display the response text incrementally in the result area.

**Figure: 5.6**

A screenshot of a computer

Description automatically generated

**Figure: 5.7**

**Program Flow and Modularity**

Upon execution, the program follows the defined flow, beginning with the initialisation of the Main class. Once instantiated, the main() method is called launching the GUI and initialising the core functionality.

While the design started as a concept with no defined plan, the distribution of classes, methods, and functions is not ideal, nor does it fully adhere to software engineering principles, which makes it difficult to draw an accurate UML diagram, hence the missing navigability markings.

However, the modularity of the design is visible, with distinct components, ensuring that the application can be easily extended and adapted to different use cases. The reliance on external classes promotes separation of concerns, making the program maintainable and scalable.

In summary, the application is a comprehensive data processing tool capable of handling multiple datasets, performing complex comparisons, and enabling intuitive interaction through both text and voice. Its modularity, ease of use, and adaptability make it an asset in research environments where efficient data analysis and interaction matter.

A noteworthy missing feature is the ability to input voice queries in languages other than English. Which was abandoned in the current iteration, while the accuracy rate of the currently used Google API is relatively high in English language comprehension, the abilities drop in accuracy when English is accented but drops to abysmal levels when tested on (Sudanese) Arabic dialect, the only other option during the research. Stanford University (2018).

**Evaluation**

Testing is critical in software development because it ensures correctness, optimises performance and enhances user satisfaction by identifying and fixing issues before the release.

Due to external limitations like tokenisation, cloud data processing costs and black box state, System Acceptance testing is done through Smoke testing and Regression testing instead of automatic testing. Whilst the program is in running order and presented as a Windows executable setup file, considering that the code is meant to match release stage practices. Unit and Integration testing has been included as samples only. \*\*\*  
The testing objective and scope are geared toward usability, correctness, efficiency, scalability and integration. The evaluation is completed against the project's initial goals as per the abstract in this research.

The project software testing strategy relied on the Viva/presentation, which should demonstrate the software's ability to perform various tasks at different difficulty levels.

Starting with base functions such as installation, GUI facilities, initiating the microphone, accepting text and voice queries, and file management. The demonstration will cover this as a necessary step before the following Functionality testing which will be completed on processing the data, processing queries, and translation to languages, including SQL.

Finally, the presentation should demonstrate the analytical abilities of the tool with difficult questions being asked, i.e. questions that would require a human to analyse the data for a long period of time.

As mentioned above, Unit Testing and Integration testing were completed at an earlier stage of the program and integral to a project of this scale, considering the 6 classes and environment files needed to complete the required functions.

With Automatic testing ruled out due to the costs associated with running the queries, the system testing required continuous Smoke and Regression testing with every improvement and feature added, considering that errors did arise whenever a new method was integrated, with some errors resulting in the removal of the features, which will be addressed in the discussions.

The final testing phase (Manual Testing) has displayed interesting results in terms of the sequence of method execution, with each method “dancing” at a different pace.

While the **main.py** is meant to control the logic of the program, each class requires a sequence of events to make it ready to integrate with the next one. On a single-class level, the program should run if there are 0 files or any number of files, with a **comparison.py** only used if two or more files are attached. Meaning that certain actions trigger other actions in sequence.

Scenarios were created to design the program functions and protocols, such as:

1. A user has uploaded 0 files and wants to query the program.
2. A user has uploaded 1 file and wants to query the files.
3. A user has uploaded 2 files and wants to query the files.
4. A user has uploaded 3 files or more and wants to query the files.

After various tests and changes, the program can easily handle the first 3 scenarios – some limitations apply – the 4th scenario is the most interesting, as the program comparison logic works best with up to 2 files. The program logic will treat the 33rd  file as the 2nd file by combining files 1 and 2 into a single Dataframe. If the 1st and 2nd files are of the same type (Tubular or text), the integration results may have negligible errors; however, if the files are of a different type, the logic will convert all files to tubular CSV, which will “ruin” text data or make it incomprehensible for the AI.

Considering that the program is only a demonstration, the above issue, while easily solved, is left to express the logic of the classes.

The **query\_handler.py** class has protocols for different situations, answering a variety of questions:

1. What if the user asks a question with or without uploading a file?
2. What if the user asks an irrelevant question to the file?
3. What if the user asks a question that is related to the previous question?

Once a file is uploaded, the query must be processed before the file in the program execution sequence, not the other way around. While this may seem “odd” when trying to limit the information scope. The rationale is that the search function needs a keyword to review the attached files.

A function to track the conversation history was created to address the 3rd point above, which an external user of a previous iteration pointed out.

The performance testing has been completed on different AI tools, while the OpenAI GPT setting, and temperature were tweaked to find the optimal configuration. Each class required suitable settings for its tasks. The **file\_handler.py** was given a low temperature to limit the possibility of hallucinations from the uploaded data, which will affect the accuracy of the answers provided. The **query\_handler.py** was given a medium temperature to allow for a creative approach to understanding the users beyond the literal voice dictation or text input. The **voice\_assistant.py** was given a high temperature, as it is responsible for generating the answers and a high temperature should accommodate for more ingenuity.

Stopping and pausing the speech function was a planned function that caused many problems and was then abandoned. Some traces of code remain not by accident but a reminder to include the function at some point.

A CMD screen appears by default setting before the GUI, displaying the background working for the program has been and still is a helpful diagnostics tool that is not suppressed by design to catch errors in real-time.

The Viva/presentation version will be running GPT-4o to demonstrate the tool's power. Not forgoing that GPT-4o is not the most cost-effective in testing and development.

|  |  |  |  |
| --- | --- | --- | --- |
| **AI Model** | **GPT-4o** | **GPT-4o-mini** |  |
| Token Limitation | Low | High |  |
| Analyses power | High | Low |  |
| Speed | Slow | Fast |  |
| Cost | High | Low |  |
| Capacity | Low | High |  |
| GPT Temperature | Scale | Scale |  |
|  |  |  |  |
| **Application Usage** | **GPT Temperature** | **Scale** | **Requirement** |
| Data Processing | 0.1 | Low | Precision |
| Query Processing | 0.5 | Medium | Interpretability |
| Generating Responses | 0.7 | High | Insightfulness |
|  |  |  |  |
| **File** | **Query** | **GPT-4o Speed** | **GPT-4o-mini Speed** |
| Small | Simple | Fast | Faster |
|  | Complex | Slow | Faster |
|  |  |  |  |
| Large | Simple | Slow | Faster |
|  | Complex | Slow | Faster |
|  |  |  |  |
| **File** | **Query** | **GPT-4o Analyses** | **GPT-4o-mini Analyses** |
| Small | Simple | Complex | Basic |
|  | Complex | Complex | Basic |
|  |  |  |  |
| Large | Simple | Complex | Basic |
|  | Complex | Complex | Basic |
|  |  |  |  |
| **File** | **Query** | **GPT-4o Cost** | **GPT-4o-mini Cost** |
| Small | Simple | Low | Lower |
|  | Complex | Low | Lower |
|  |  |  |  |
| Large | Simple | High | Low |
|  | Complex | High | Low |
|  |  |  |  |
| **File** | **Query** | **GPT-4o Capacity** | **GPT-4o-mini Capacity** |
| Small | Simple | High | High |
|  | Complex | High | High |
|  |  |  |  |
| Large | Simple | Low | High |
|  | Complex | Low | High |

**Table 1**

The adjacent table is created to express the differences between GPT-4o and GPT-4o-mini, for which the latter was used extensively throughout the development and testing.

There are some concerns with the design, as in the GPT model selection and temperature and outcome factoring. Where a different setting or combination of AI models can be more efficient in terms of resource use and more accurate in terms of responses.

The information about the GPT model performance is based on OpenAI Literature and real-world testing. OpenAI. (n.d.). GPT-4o. In OpenAI API. https://platform.openai.com/docs/models

The system has been tested on all the listed file types (Excel, CSV, PDF, DOCX, and SQL), while other file types were tested for error handling.

The most complex testing takes place with tubular file formats, as this is how databases are set in most cases. The ability to extract, load, navigate, validate, visualise, and manipulate the data stored in these databases.

The ability to analyse the data and generate useful insights that will have taken a human longer to create or achieve.

On a dataset of Covid-19, the following questions were asked:

1. Which country has the highest mortality rate and why?
2. Which 3 countries are reporting the lowest numbers and why?
3. Which 5 countries have the highest death rate and why?

With most answers being arguably reasonable, acceptable and verified, and would have taken a human a long time.

**Discussion**

A major concern in this program's inception was managing live databases, not converted SQL files and the currently supported file formats. However, not having access to a live database was expected due to legal and privacy concerns with using such a tool.

The live database owners' concerns are not limited to allowing a single individual access, but the issue extends further to the servers processing the data and queries. As there are concerns relating to Privacy and Data Retention. (Zaman, 2023)

During the research, interesting tools were found that could be integrated with the program, most specifically (Unstructured & Amazon AWS) libraries and API’s. Ultimately, creating an online storage unit where all the files are deposited and accessed from any device with the help of AWS, with Unstructured building the dataframes with higher file handling capacity, especially the ability to deconstruct image and audio files. Unstructured.io (2023, September 8)

Costs incurred while using a prepaid API key depend on the coded LLM model, for which the process can be optimised further with a class dedicated to deciding which GPT model is most suitable to handle the query, response and/or files. Furthermore, the temperature can also be adjusted accordingly depending on the inputs and the expected output. As a result, only one class uses the top-of-the-line LLM model, while the rest is made of variables created by the model and temperature-deciding class. Furthermore, the modularity can help add an image analysis facility (without Unstructured) with the currently available image analysing and generating LLM model. i.e. (Google’s PaLM-E and OpenAI’s DALL·E)

Recursive self-improvement is an area of research that is worth exploring with this software. The current program can analyse its own Python coding when presented as a document file, for which it can locate bugs and suggest improvements. The question is: What will happen if a self-improvement loop is established? where the program will analyse itself, change and add features automatically.

**Conclusion**

The program has arguably met its initial design concept with negligible errors or bugs. The graphic design improvements would have required a time investment that is a trade-off with adding features and/or improving the logic performance. However, the one-tool solution to databases is somewhat possible and could be achieved at a commercial level with the right amount of investment.

A fundamental discussion point is the HCI element of software development, especially in database design. Projects must balance development and testing with release usability. Developers will get used to testing the design in a certain manner that will require a usage manual or training for a new user. External feedback from blind testers did provide feedback guiding the development of the GUI and features.

The development process proved that the unappealing GUIs we see result from the development and testing phase, as happened during this code development. However, good design was the initial purpose of the project, for which, even while missing the target, the results were arguably satisfactory.

Potential

**References**

Admin Matters: The Impact Of NHS Administration On Patient Care. (n.d.). The King’s Fund. https://www.kingsfund.org.uk/insight-and-analysis/long-reads/admin-matters-nhs-patient-care

Austine, O., Olalekan, M., Babatunde, Y., & Lateef, O. (2023). The Integration Of Artificial Intelligence Into Database Systems (Ai-Db Integration Review).

Bohr, A., & Memarzadeh, K. (2020). The rise of artificial intelligence in healthcare applications. In Artificial Intelligence in Healthcare (pp. 25–60). Elsevier. https://doi.org/10.1016/B978-0-12-818438-7.00002-2

Ghai, W., & Singh, N. (2012). Literature Review on Automatic Speech Recognition. International Journal of Computer Applications, 41(8), 42–50. https://doi.org/10.5120/5565-7646

Ioannidis, Y. E. (1996). Visual user interfaces for database systems. ACM Computing Surveys, 28(4es), 137-es. https://doi.org/10.1145/242224.242399

Kelion, L. (2020, October 5). Excel: Why using Microsoft’s tool caused Covid-19 results to be lost. BBC News. https://www.bbc.co.uk/news/technology-54423988

Li, G., Zhou, X., & Cao, L. (2021a). AI Meets Database: AI4DB and DB4AI. Proceedings of the 2021 International Conference on Management of Data, 2859–2866. https://doi.org/10.1145/3448016.3457542

Li, G., Zhou, X., & Cao, L. (2021b). Machine Learning for Databases. The First International Conference on AI-ML-Systems, 1–2. https://doi.org/10.1145/3486001.3486248

Li, G., Zhou, X., & Li, S. (n.d.). XuanYuan: An AI-Native Database.

McCarthy, D. R. (n.d.). The architecture of an active database management system.

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: An introduction. Journal of the American Medical Informatics Association, 18(5), 544–551. https://doi.org/10.1136/amiajnl-2011-000464

OpenAI. (n.d.). GPT-4o. In *OpenAI API*. Retrieved from <https://platform.openai.com/docs/models>

Zaman, S. (2023). ChatGPT security risks and solutions. *7th IET Smart Cities Symposium (SCS2023)*, 378–387. https://doi.org/10.1049/icp.2024.0955

Unstructured's Commercial SaaS API. [Unstructured.io](https://unstructured.io/docs/commercial-saas-api/)

Stanford University (2012). Speech Recognition for Accented English [Report]. Retrieved from <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/reports/custom/report24.pdf>

**Appendices**

**Tools & Libraries, and Descriptions of Respective Functions**

- tkinter: Used to create the main application window, buttons, text fields, and other GUI components.

- threading: Allows the application to perform tasks in the background, such as voice recognition and text-to-speech, without freezing the GUI.

- time: Used to introduce delays and manage timing for certain actions.

- re: Utilized for regular expression operations, such as checking and correcting text input.

- os: Provides a way to interact with the file system.

- io: Used for handling input/output operations, especially when dealing with files.

- pandas: Essential for loading, manipulating, and analysing tabular data from various file formats.

- docx: Enables reading and extracting text from Microsoft Word documents.

- pdfplumber: Facilitates extracting text and tables from PDF files.

- sqlalchemy: Provides tools for database interaction, including executing SQL queries and managing database connections.

- langchain\_openai: Enables interaction with OpenAI's for generating responses to queries.

- dotenv: Loads API keys and other configuration settings from a .env file.

- speech\_recognition: Captures and converts voice input into text.

- pyttsx3: Converts text responses into speech, allowing the application to "speak" to the user.

- spellchecker: Checks and corrects the spelling of user queries.

- filedialog: Opens dialog windows for file selection.

- messagebox: Displays informational, warning, and error messages to the user.

- scrolledtext: Provides a text area with a scrollbar for displaying long responses.

- tkfont: Manages font styles and sizes for the GUI components.

- file\_handler: Custom module for handling file operations, including loading data from various file formats.

- comparison: Custom module for comparing multiple DataFrames and summarizing their differences.

- gui\_handler: Custom module for managing GUI interactions and updating the interface based on user actions.

- query\_handler: Custom module for processing user queries and generating responses using the language model.

- voice\_assistant: Custom module for handling voice input and output, including capturing voice queries and generating spoken responses.

- Microsoft C++ Build Tools: Required to develop Windows Applications

**Code snippets**

**comparison.py**

*Compare multiple DataFrames and return a summary of their differences.*

def \_compare\_multiple\_dataframes(self) -> str:

comparison\_results: List[str] = []

num\_dfs = len(self.dataframes)

for i in range(num\_dfs):

for j in range(i + 1, num\_dfs):

df1 = self.dataframes[i]

df2 = self.dataframes[j]

comparison\_results.append(f"Comparing DataFrame {i + 1} with DataFrame {j + 1}:")

comparison\_results.append(self.\_compare\_two\_dataframes(df1, df2))

comparison\_results.append("\n") # Add a newline between comparisons

return "\n".join(comparison\_results)

**file\_handler.py**

*Generate and execute SQL for database management tasks.*

    def manage\_database(self, management\_prompt: str) -> str:

        if not self.engine:

            raise ValueError("No database connection available for management.")

        sql\_code = self.generate\_sql(management\_prompt)

        try:

            with self.engine.connect() as conn:

                conn.execute(text(sql\_code))

            return f"Executed SQL: {sql\_code}"

        except Exception as e:

            return f"Error executing SQL: {str(e)}"

    def \_update\_comparison\_summary(self) -> None:

**query\_handler.py**

*Create a prompt based on the content type, question, and conversation history.*

def \_create\_prompt(self, content: Union[pd.DataFrame, str], question: str) -> str:

        """

        Create a prompt based on the content type, question, and conversation history.

        Args:

            content (Union[pd.DataFrame, str]): The content to base the answer on.

            question (str): The question to be answered.

        Returns:

            str: The generated prompt.

        """

        if isinstance(content, pd.DataFrame):

            content\_str = content.head(250).to\_string(index=False)

            content\_str = f"DataFrame content:\n{content\_str}\n\n"

        else:

            content\_str = f"Comparison Summary:\n{content}\n\n"

        # Combine history into prompt, only including the latest context

        history\_str = "\n\n".join(self.history)

        return f"{history\_str}\n{content\_str}Question: {question}\nAnswer:"