Enhancing Data Accessibility: Integrating Speech-Based Interfaces with Large Language Models for Intuitive Database Queries

Abdullah Kiwan[1], Andreas Lommatzsch[2] and Sahin Albayrak[1,2]

[1]*GT-ARC, TU Berlin, Ernst-Reuter-Platz 7, D-10587 Berlin, Germany*
[2]*DAI-Labor, TU Berlin, Ernst-Reuter-Platz 7, D-10587 Berlin, Germany*

Abstract
Accessing complex database information often requires specialized knowledge. Existing dashboard tools supporting only pre-defined queries are inflexible and inadequate for many users. This paper introduces a novel approach by combining an intuitive, speech-based interface with a Large Language Model (LLM) specifically trained to understand database structures and generate user-friendly answers and visualizations. Our method focuses on simplifying the interaction for non-expert users by allowing them to ask questions directly and receive insights without the need for database engineers or data scientists. This is especially important to handle follow-up questions raised by anomalies in generated answers. We present a detailed analysis and discussion of a use case that demonstrates the practicality and effectiveness of our approach through a developed prototype. We study the strengths and weaknesses of the system components as well as the user feedback for the system. Based on the observations further research directions are discussed.

1. Introduction

In today's data-driven world, the need for efficient and user-friendly access to information stored within complex databases is more critical than ever. Traditional systems often rely on predefined dashboards and queries, which can be inflexible and inadequate for users who require dynamic access to data. The implementation of new queries often requires database engineers or data scientists, making the process inefficient and slow for users without technical expertise.

To address these challenges, we propose an innovative approach that merges the intuitiveness of a speech-based interface with the robust capabilities of a Large Language Model (LLM). Our idea focuses on creating an LLM that is not only trained to understand and generate human-like responses but also equipped to comprehend the underlying structure of a database. By teaching the model typical user inquiries and dialogues, we aim to empower normal users to retrieve complex data and generate visual representations independently, without the need for specialized knowledge.

In the evaluation of the developed system, we focus on the following research questions:

abdullah.kiwan@gt-arc.com (A. Kiwan); andreas.lommatzsch@dai-labor.de (A. Lommatzsch); sahin.albayrak@dai-labor.de (S. Albayrak)

- How well current LLMs perform without post-training/fine-tuning in the given scenario?
- How reliable company specific questions can be answered by our system (considering domain specific vocabulary and the probability of incorrect answers)?
- What are the optimal strategies for achieving high levels of user satisfaction? What are the performance limitations of the LLM when processing large and complex datasets?
- What are the potential areas for further research to improve the capabilities of the application?

Our contribution to this field includes a comprehensive analysis and discussion of a case that illustrates the practical challenges and demands of current information retrieval systems. We have developed a prototype that converts user speech into SQL-based database queries and generates answers including diagrams answering the user's information demand. Based on the analysis of the strengths and weaknesses of our prototype we discuss the research needs.

The remainder of this paper is organized as follows. Sec. 2 describes the problem in detail. Sect. 3 discusses existing methods to identify promising methods and strategies. Sec. 4 describes the developed system and the pipeline deployed for generating answers to the concrete information demand. Subsequently, the evaluation of our approach is presented in Sec. 6 analyzing the results from a quantitative and qualitative perspective. Finally, Section 7 discusses the generalization of our findings and Sec. 8 provides an outlook on future work.

2. Problem Description

In today's business world, having quick access to important information stored in complex databases is crucial. However, the current methods of getting this information are often slow and inefficient. Employees typically have to wait for database engineers and data analysts to generate reports, which can delay decision-making and reduce productivity.

While data analytics tools (like Microsoft Power BI[1] or Tableau[2]) exist to help create reports from databases, they have limitations. These dashboards rely on predefined queries and require users to have some technical knowledge to use them effectively. This makes it challenging for employees without specialized skills to get the data they need quickly. Furthermore, predefined queries do not offer the flexibility to answer varied and dynamic questions. Users often end up with static reports that do not fully address their specific needs, leading to repeated requests for new reports.

To solve these problems, there is a need for an application that allows users to ask questions in natural language and get immediate answers from the database. In addition, follow-up questions should be supported to provide deeper insight into specific questions. A natural language interface would allow employees to quickly access the information they need without relying on technical experts. An assistant that understands the user's intentions and database structures could provide more intuitive and flexible interactions with complex data. Such a solution would empower users to retrieve and interpret data independently, leading to better and faster decision making and a more efficient business environment.

[1]https://www.microsoft.com/de-de/power-platform/products/power-bi
[2]https://www.tableau.com

In our approach, we consider several user-related as well as technical challenges we discuss in subsequent subsections.

2.1. User-Related Challenges

To ensure the user satisfaction, the user expectations must be considered.

Understanding User Expectations One of the primary challenges is aligning our understanding with the customers' expectations. Users, particularly those without a technical background, have very diverse beliefs about LLMs, often expecting that an LLM can read the user's mind and grasp their intent without clear and precise input. This misunderstanding necessitated thorough communication regarding the capabilities of LLMs, and required methods to guide and support users to formulate questions that contain all relevant information and demands to achieve the desired outcomes.

Vocabulary and Synonyms Each organization employs specific terminology that is unique to their internal operations. It is a significant challenge to ensure that LLMs understands and correctly interpreted these company specific terms. It requires close collaboration with the customers to compile a comprehensive list of synonyms and commonly used phrases. Integrating this vocabulary into the LLM is essential for tuning the model to respond accurately to user queries.

2.2. Technical Challenges

In addition to the user-related challenges, the technical challenges must be addressed.

Hallucination Systems making use of LLMs have to fight with the phenomenon of hallucination, where the LLM generates plausibly sounding, but inconsistent or incorrect answers. This issue is particularly apparent during the evaluation process, as it occasionally produced varying results for the same input. Addressing this instability is critical to ensure the reliability of the application.

Query Optimization The LLM may generate code for querying data that is executable, but not optimized resulting in query timeouts. This occurs for complex information demands requiring multiple database table join operations. Optimizing the generated queries to handle complex database structures effectively is a challenging aspect in the development process that should be addressed by the database management software and the module that generates the query.

Performance and Speed A long response time of web applications is a frequently reported problem from the user perspective. The latency primarily stemmed the chain of several LLM's prompts required for generating the SQL query and "transforming" the database response into natural language answers or diagram. The dual reliance on the LLM contributed to delays, which often impacts the user satisfaction. Identifying and mitigating these performance bottlenecks and paralleling steps is important to enhance the user experience.

Developing a solution that addresses these user and technical challenges is key to ensuring an efficient, robust application that provides intuitive access to complex database information and simplifies business decision making.

3. Related Work

In this section we review existing tools and discuss existing approaches.

3.1. Data Analysis, Visualization, and Business Intelligence Tools

The landscape of data analysis and visualization has evolved significantly, with various tools emerging to meet the diverse needs of businesses and analysts. This subsection discusses some of the leading tools in the market.

Microsoft Power BI[3], Looker Studio[4],and Tableau[5] are business intelligence tools designed to provide interactive visualizations and business intelligence capabilities. They support a wide range of data sources and offer features such as dashboards, reports, and integration with other tools. These tools are designed for data analysts; based on low code-based user interfaces also non experts should be able to create queries. Recent version of Microsoft Power BI and Tableau provide a component for deriving simple query from natural language inputs by detecting entities in unstructured texts.

For visualization, NL4DV [1] is a toolkit for natural language-driven data visualization. It takes as input a tabular dataset and a natural language query about that dataset. In response, it returns an analytic specification.

The analysis shows that there is a need for simplifying the access to business intelligence. The existing systems try to add additional components, but the focus is still on extended support for data scientists. In our work, we start from a chat perspective. Thus, users can benefit from all the strengths of Large Language Models (wide range of questions, dialog-style follow up questions) and enter database-related questions in an intuitive way (using speech or natural language).

3.2. Pretrained Models for Text-to-SQL

The generation of SQL queries based on unstructured texts is an vivid research topic. Several models and approaches have been suggested:
SQLNet is a model developed to address the "order-matters" problem in SQL query generation by employing a sequence-to-set model and a column attention mechanism [2]. **Seq2SQL** is a deep neural network for translating natural language questions to corresponding SQL queries [3]. The model uses rewards from in-the-loop query execution over the database to learn a policy to generate the query.
SQLova is a powerful text to SQL model to achieve a high accuracy in WikiSQL dataset [4]. It uses table-aware word contextualization with large pre-trained language model BERT [5].

The analysis of these models shows that different methods for generating queries based on unstructured user inputs exist. Fine-tuning of the models for concrete databases and scenarios strongly improves the performance. Due to the fast changes in the domain of Large Language Models the results are often difficult to reproduce. In our research we focus on zero shot

[3]https://www.microsoft.com/de-de/power-platform/products/power-bi
[4]https://cloud.google.com/looker-studio
[5]https://www.tableau.com

learning. We use standard Large Language Models to ensure that the models are not limited to query generation and can be updated easily. We analyze how successful non-tuned LLMs can be used for generating answers based on business databases.

3.3. Large Language Models

In addition to pre-trained models for text-to-SQL there are several interesting approaches that extend the idea of Large Language Models: **PICARD** [6] introduces a novel approach to text-to-SQL generation by constraining auto-regressive decoders of language models through incremental parsing. **EPI-SQL** [7] is a framework leveraging Large Language Models to enhance the performance of text-to-SQL tasks. EPI-SQL involves collecting instances from the dataset where LLMs fail. These instances are then utilized to generate general error-prevention instructions (EPIs). **SQL-PaLM** [8] extends the capabilities of large language models by applying few-shot prompting and instruction fine-tuning. **SGU-SQL** [9] is a structure-to-SQL framework, which leverages the inherent structure information to improve the SQL generation of LLMs. **DTS-SQL** [10] is a novel approach that decomposes the task into two simpler tasks solved by a small LLM. **PET-SQL** [11] introduces a two-round framework to enhance the performance of LLMs based on few-shot prompting and cross-consistency across different LLMs. The performance of the models can be evaluated using the frameworks by Rajkumar et al. [12] or the methods by Roberson et al. [13].

The analyzed systems suggest strategies for SQL generation by using incremental parsing, few-shot prompting, instruction fine-tuning, and structural information leveraging. These systems show a high complexity and resource demands. In our work we try to keep the complexity low to ensure extensibility and adaptability. Our approach is implemented in a real corporate setting, contrasting with the theoretical or lab-based studies often cited. We use it with real users and in daily operations, providing practical insights that set our work apart from the more experimental studies in existing papers.

4. Approach

We develop a chatbot prototype to provide answers (including graphical visualizations) for natural language questions. We use a flexible, component-based approach enabling the integration and evaluation of NLP- and AI algorithms. The architecture of our system is shown in Figure 1. The typical processing pipeline and the specific capabilities of the developed component are explained in the following sections.

4.1. Database Query Builder

The core component of our system is the query builder designed to derive database queries from natural language text inputs. The component provides APIs to integrate existing Large Language Models, such as OpenAI GPT-4 [14] or a locally hosted Llama-3 [15] model. We start with existing LLMs and provide all application specific information in the system prompt. The system prompt includes the data structure to ensure that the database column name, the date types and the reference relation between the tables are known to the LLM.

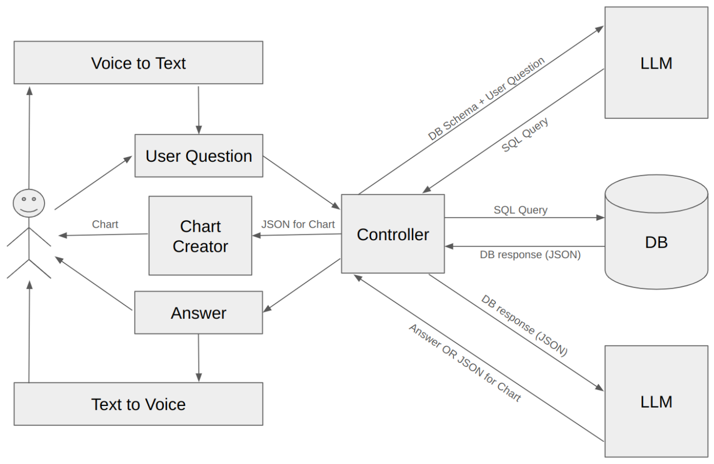


Figure 1: *The System Architecture*

In addition, application specific vocabulary and synonyms are provided in the system prompt making the mapping from "colloquial" user inputs to the formal database model more robust and reliable. The used approach requires additional discussions with the users and domain experts; but standard LLMs do not know niche application user vocabularies well enough for ensuring high quality data query results. In the design process we discussed how to teach the domain specific synonyms in the LLM-fine tuning process. we opted for using the system prompt enabling us to integrate different LLMs. This also simplifies the use of recent, more powerful LLMs.

4.2. Database Connector

The query generated by the query builder is sent to the database. The retrieved data are formatted as json files simplifying the post processing of the data. The database connector gives access to different databases, using standard connectors, such as ODBC. The database connection should ensure that the SQL-dialect is supported as well as that large query outputs are processed efficiently (e.g. by setting limits for the output size). This is particularly important to ensure the rapid responses that are the basis of a good user experience.

4.3. Processing the Database Results

After retrieving the data required to satisfy the user's information needs, the data is transformed into either a natural user response or a diagram that visually provides the answer.

Text Output Component If the user requests the output in plain text format, a dedicated LLM agent processes the data and generates a coherent and concise textual response. This agent ensures that the information is presented clearly and accurately, making it easy for the user to understand.

Visualization Output Component If the user prefers a visual representation of the data, another LLM agent is responsible for creating appropriate charts or graphs. This agent interprets the data and selects the most suitable visualization format, enhancing the user's ability to grasp complex information quickly and effectively.

4.4. Multi-Modal User Interface (Chatbot-Based Interface)

At the core of the application is a chatbot-based interface, providing an intuitive and user-friendly platform for interacting with the database. This design choice ensures that users can engage with the system using natural language, facilitating a more accessible and efficient means of querying and retrieving data without requiring specialized technical knowledge.

Voice-to-Text Support The speech-to-text feature in our application enhances user interaction by allowing users to verbally communicate their queries, which are then converted to text for further processing. This functionality is implemented using JavaScript libraries that make use of the browser's built-in capabilities. The Web Speech API's SpeechRecognition[6] interface provides a powerful mechanism for capturing and transcribing speech in real-time ensures a seamless user experience.

Text-to-Voice Functionality The text to speech feature complements the speech to text functionality by enabling the application to provide spoken responses to user queries. This feature is particularly useful for users who prefer auditory feedback or have visual impairments. Our system also uses the Web Speech API's SpeechSynthesis[7] interface.
In order to process a user request, several calls of the LLMs are required, resulting in a chain of LLM-calls and an interaction with the database. Each LLM call and database output for generated queries may produce unexpected results (e.g. unexpected data format or timeout) requiring an exception handling. Specific exceptions, such as invalid data formats returned by the LLM, can often be resolved by retrying the call. If the pipeline fails, the user must be notified, as non-technical users are often sensitive to unexpected results. Since LLMs are still less reliable compared with traditional methods, users should be aware of the fact, that not all questions can be answered in the first try.

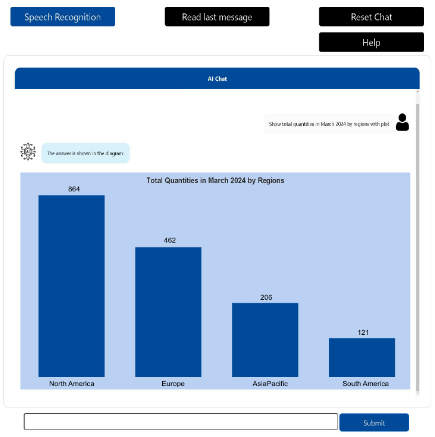
5. Implementation and Deployment

Our prototype has been implemented based on a Python-based backend and a vue.js-based frontend. It has been deployed on a Ubuntu Linux-based virtual machine using docker containers.

The User Interface

The user interface is developed as a modern web-based UI. The UI allows the user to interact in a text chat. In addition, users can enter speech; generated answers can be read by the system by integrating a text-to-speech module. Figure 2 shows an example of the UI.

[6]https://developer.mozilla.org/en-US/docs/Web/API/SpeechRecognition
[7]https://developer.mozilla.org/en-US/docs/Web/API/SpeechSynthesis



Show total quantities in March 2024 by regions with plot

The answer is shown in the diagram:

Figure 2: *The chatbot window supports text and speech inputs. Users get text-based answers and diagrams answering the given question.*

Backend Implementation

The backend has been developed using Python, leveraging LangChain for the LLM and chatbot functionalities. LangChain facilitates the integration of advanced natural language processing capabilities, enabling the system to effectively understand and respond to user queries.

Database Integration To connect our application to the client's database, a dedicated virtual machine was provisioned. This virtual machine served as the deployment environment for the application, ensuring secure and efficient access to the database management system.

The application was deployed on the virtual machine using Docker, which provided an isolated and consistent environment for running the application. Docker simplifies the process of creating, deploying, and running applications in containers, ensuring that the application performs reliably regardless of the underlying infrastructure.

The connection to the database management system was established using pymssql, a Python library that facilitates interaction with Microsoft SQL Server databases. This setup allowed the application to execute SQL queries and retrieve data according to user requests.

Integration of LLMs We integrated the Azure API to connect and utilize the GPT-4 [14] model. The Azure API facilitated seamless access to GPT-4 [14], ensuring that our application could utilize the latest advancements in natural language processing without the need for extensive on-premises infrastructure.

Conclusion The integration of LangChain, Docker, and the APIs to recent LLMs into our back-end architecture not only streamlines the development and deployment processes but also enhances the system's capability to deliver robust and high quality responses tailored to user interactions. The used technical basis of our application ensures an easy deployment and the flexibility to integrate new models and components.

6. Evaluation

The developed prototype has been evaluated in a real-world setting within a company, providing us with access to live data. Employees are asked to test the system to evaluate the performance, usability and perceived benefits.

6.1. Evaluation Metrics and Benchmarking

To assess the performance of our system, employees provided a set of frequently asked questions that they anticipate using regularly. We utilized these questions as benchmarks, categorizing them into three levels of difficulty: Easy, Medium, and Hard.

- *Easy* (3 questions): This level includes questions involving basic database operations such as a sum operation, group by, and a single join. An example of such a question is: "Show total quantities by region on May 14, 2024."
- *Medium* (9 questions): Questions in this category are more complex, incorporating operations similar to those in the Easy category but require two join operations. For instance: "Show total quantities by region on May 14, 2024, for the TIRE industry."
- *Hard* (3 questions): This category comprises questions similar to Medium but requiring additional reasoning to translate customized information in the prompt into an SQL query. An example is: "Show total quantities in March according to BIG 6," where "BIG 6" refers to a specific group of customers defined in the database. The calculation method for BIG 6 is provided in the prompt, and the LLM is expected to incorporate this into the SQL query effectively.

To measure the success of each question within our evaluation framework, we established a specific metric based on the accuracy and functionality of the SQL queries generated by the LLM. A question is considered successfully answered if it results in a SQL query that is not only syntactically correct but also executes without errors and retrieves the correct response from the database.

Each question within this benchmark can be varied significantly by changing elements such as the date, date range, or specific products, thereby make more queries for our system to evaluate. The evaluation shows that while the Easy questions were successfully processed, the Medium and Hard questions required additional tailored information in the prompt to be successful. For instance, to accurately compute 'Big 6', specific details about the necessary database table and column needs to be included in the prompt.

6.2. Synonyms, Vocabulary, and Hallucination

At the outset of the project, company employees provided us with the set of benchmark questions and the corresponding correct SQL queries. Initially, these questions were used to test the system without integrating the company's specific synonyms and vocabulary into the LLM. The results of these early tests were successful for the Easy questions, but unsatisfactory for the Medium and Hard questions; the LLM often failed to generate correct SQL queries, leading to incorrect results or errors during the execution of the SQL queries on the database.

A significant issue observed during this phase was the phenomenon of hallucination, where the LLM produced unstable and varying SQL queries for the same input. This instability was a direct consequence of the model's lack of familiarity with the specific terminology used within the company.

To address these problems, we integrated the company-specific synonyms and vocabulary into the LLM. This enhancement resulted in a marked improvement in system performance. The integration eliminated hallucination and significantly stabilized the generation of SQL queries. Post-integration tests showed that the LLM was able to consistently generate correct and reliable SQL queries thus ensuring accurate results and reducing execution errors.

6.3. Analyzing Llama 3 and GPT-4 for Text to Query Generation

Our system enables the flexible integration of different LLMs. In our tests, we conducted a comparative analysis between Llama 3 [15], an open-source large language model, and GPT-4 [14], a closed-source model, specifically for their performance in text-to-query generation.

We tested both models using the set of questions provided by the client. The test set included questions of different complexity, starting with questions that could be answered based on one database table to questions that required multiple join-operations. The results indicated a significant performance difference between the two models:

GPT-4 [14] consistently produced more accurate and optimized SQL queries. The queries generated by GPT-4 [14] successfully retrieved the correct data and did not result in any errors. The model demonstrated a robust understanding of the database structure and user-specific vocabulary, leading to precise and reliable outputs. Even complex join-operation have been generated correctly.

In contrast, **Llama 3** [15] exhibited several limitations. A majority of the queries generated by Llama 3 [15] resulted in timeout errors, indicating that the model struggled to produce optimized queries. Generated SQL-queries contained often more join-operations than needed. Thus, the SQL queries for simple questions worked OK, but for more complex questions the generated SQL queries resulted in a timeout. In addition, in about 25% of the cases Llama 3 generated queries contained syntax errors for accessing database columns. Thus, errors are easily to fix for normal database users, but for non-experts, even simple errors are a problem.

In conclusion, GPT-4's [14] performance in generating accurate and efficient SQL queries far surpasses that of Llama 3 [15]. GPT-4 [14] reliably generated SQL queries containing up to three join operations. On the other hand, open-source models like Llama 3 [15] offer flexibility and accessibility enabling an scenario-optimized fine-tuning. This means, that the integration of Llama 3 [15] needs additional efforts and cost to optimize the LLM - but it also gives a higher level of data security and the opportunity to do fine-tuning. As discussed in the related work section, several projects exist that fine-tune Llama for the generation of SQL queries to the performance is comparable to GPT-4 [14]. This means, that GPT-4 [14] performs very well in our scenario without any adaptation; but based on the enterprise guidelines and available capacities, open-source models such as Llama 3 [15] can be considered if enough resources for fine-tuning are available.

6.4. User Impressions

As an initial evaluation phase, we had 3 users, a Data Engineer, and two sales people since the application will be used in the sales department. Upon finalizing the application, it was subjected to extensive testing by the company's employees. Overall, the feedback from users was positive. Users appreciated the system's ability to understand and respond to their queries accurately. The stability and correctness of the SQL query generation, post-synonym integration, were particularly well-received. Since the system is based on a general purpose LLM, the system does not only answers database related questions, but it also provides answers for a wide spectrum of questions.

However, users raised concerns about the response time of the application. While the system's performance in terms of accuracy and stability was satisfactory, the time taken to generate and return the final output was longer than desired. For some queries, the response time was as high as 15 seconds. Users expressed a preference for a maximum response time of 5 seconds, indicating a need for further optimization in terms of processing speed. This is a challenging demand, since the time need by LLM for generating answers depends on the amount of input data. Providing complex data base schema results in a slow answer behavior.

Discussion Through our evaluations, several key findings emerged:

- Effective communication with users about the capabilities and limitations of the LLM is crucial for setting realistic expectations and achieving accurate results.
- Incorporating user-specific vocabulary and synonyms into the LLM significantly improves its performance and accuracy. Managing LLM-hallucination and ensuring stable outputs require ongoing evaluation and refinement of the model. An extended error handling is crucial to provide well readable answers even if a step in the processing pipeline fails or must be repeated due to unstable LLM answer behavior.
- The speech-to-text and text-to-speech features work well in demonstration settings. Even though speech support is technically not hard, it gives users the impression to interact with an intelligent system. A weakness is the addition processing time - in a speech-based dialog fast responses are expected by most users.
- Improving the speed of the application is vital to user satisfaction, which requires efforts to streamline the LLM processes involved in query generation and response formulation.

Overall, our test users have been satisfied with the prototype since it handled the question set (defined by the test users) well. Still there is room for improvement especially with respect to runtime and handling complex questions.

7. Conclusion

This study addresses the critical need for more flexible and accessible methods to extract information from complex databases, highlighting the limitations of traditional dashboards and the dependency on technical experts. Our approach combines a speech-based interface with a LLM, specifically trained to understand database structures, generate insightful responses and provide visualizations. The developed prototype demonstrates the potential in simplifying data retrieval and interaction for non-expert users, thereby democratizing data access.

Our analysis of the developed prototype demonstrates that Large Language Models, such as GPT-4, excel at producing accurate and efficient SQL queries, even without fine-tuning. Adequate prompting (incorporating domain-specific vocabulary) is sufficient to achieve this performance. The queries generated under these conditions are both precise and dependable, providing reliable answers.

8. Future Work

The evaluation results show that there is still room for several improvements and extensions.

Multi-Agent LLM System The current system uses only one LLM for handling user queries. In order to improve the precision and the performance of the system, several task-optimized ("fine-tuned") LLMs could be integrated using a multi-agent approach. We expect that this would speed up the user input processing and improve the response quality, but would lead to a higher technical complexity and additional effort for tuning the integrated LLMs.

Enhanced Query Optimization When users tested our system, they typically started with simple questions that could be answered based on one or two database tables. This motivates the users to test more complex questions requiring multi-join queries. Such queries may run into timeout due to complex join operations. When debugging such cases, we manually tuned the queries by reordering and limiting the operations in generated SQL queries. In the future this optimization step should be performed with an AI-based component trained on examples optimized for the used database engine and the relevant SQL dialect. In addition, the system could detect very complex user demands and suggest simpler, but still useful questions.

Detecting and Handling Hallucinations Invalid answers due to incorrect SQL queries are a big challenge. It is for technically inexperienced users very difficult to check the plausibility of answers. Thus, the system should provide strategies for checking and explaining the reliability of answers. To cope with hallucinations, we plan to explore new training methodologies that reduce the likelihood of hallucinations. Furthermore, we plan to utilizing feedback loops to correct inaccuracies in real-time, thereby improving the model's reliability and user trust over time. In addition, we will explore additional validation components to ensure the accuracy and reliability of the generated responses.

By pursuing these research directions, we aim to build on the foundation of our current application to create a more robust, efficient, and user-friendly tool for the interaction with databases. These efforts will contribute to the advances in fields of natural language processing and database management, ultimately leading to more accessible and powerful data analysis solutions for users across various of industries.

Acknowledgment

This paper is based on the research conducted in the Go-KI project (Offenes Innovationslabor KI zur Förderung gemeinwohlorientierter KI-Anwendungen) funded by the German Federal Ministry of Labour and Social Affairs (BMAS) under the reference number DKI.00.00032.21.

References

[1] A. Narechania, A. Srinivasan, J. Stasko, NL4DV: A Toolkit for Generating Analytic Specifications for Data Visualization from Natural Language Queries, IEEE Transactions on Visualization and Computer Graphics 27 (2021) 369–379. URL: http://dx.doi.org/10.1109/TVCG.2020.3030378. doi:10.1109/tvcg.2020.3030378.

[2] X. Xu, C. Liu, D. Song, SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning, 2017. URL: https://arxiv.org/abs/1711.04436. arXiv:1711.04436.

[3] V. Zhong, C. Xiong, R. Socher, Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning, 2017. URL: https://arxiv.org/abs/1709.00103. arXiv:1709.00103.

[4] W. Hwang, J. Yim, S. Park, M. Seo, A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization, 2019. URL: https://arxiv.org/abs/1902.01069. arXiv:1902.01069.

[5] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, in: J. Burstein, C. Doran, T. Solorio (Eds.), Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186. URL: https://aclanthology.org/N19-1423. doi:10.18653/v1/N19-1423.

[6] T. Scholak, N. Schucher, D. Bahdanau, PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models, in: M.-F. Moens, X. Huang, L. Specia, S. W.-t. Yih (Eds.), Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 2021, pp. 9895–9901. URL: https://aclanthology.org/2021.emnlp-main.779. doi:10.18653/v1/2021.emnlp-main.779.

[7] X. Liu, Z. Tan, EPI-SQL: Enhancing Text-to-SQL Translation with Error-Prevention Instructions, 2024. URL: https://arxiv.org/abs/2404.14453. arXiv:2404.14453.

[8] R. Sun, S. Ö. Arik, A. Muzio, L. Miculicich, S. Gundabathula, P. Yin, H. Dai, H. Nakhost, R. Sinha, Z. Wang, T. Pfister, SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL (extended), 2024. URL: https://arxiv.org/abs/2306.00739. arXiv:2306.00739.

[9] Q. Zhang, J. Dong, H. Chen, W. Li, F. Huang, X. Huang, Structure Guided Large Language Model for SQL Generation, 2024. URL: https://arxiv.org/abs/2402.13284. arXiv:2402.13284.

[10] M. Pourreza, D. Rafiei, DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models, 2024. URL: https://arxiv.org/abs/2402.01117. arXiv:2402.01117.

[11] Z. Li, X. Wang, J. Zhao, S. Yang, G. Du, X. Hu, B. Zhang, Y. Ye, Z. Li, R. Zhao, H. Mao, PET-SQL: A Prompt-Enhanced Two-Round Refinement of Text-to-SQL with Cross-consistency, 2024. URL: https://arxiv.org/abs/2403.09732. arXiv:2403.09732.

[12] N. Rajkumar, R. Li, D. Bahdanau, Evaluating the Text-to-SQL Capabilities of Large Language Models, 2022. URL: https://arxiv.org/abs/2204.00498. arXiv:2204.00498.

[13] R. Roberson, G. Kaki, A. Trivedi, Analyzing the Effectiveness of Large Language Models on Text-to-SQL Synthesis, 2024. URL: https://arxiv.org/abs/2401.12379.

arXiv:2401.12379.

[14] OpenAI, The GPT-4 Team, GPT-4 Technical Report, 2024. URL: https://arxiv.org/abs/2303.08774. arXiv:2303.08774.

[15] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, LLaMA: Open and Efficient Foundation Language Models, 2023. URL: https://arxiv.org/abs/2302.13971. arXiv:2302.13971.