Generative AI based Virtual Assistant for Reconciliation Research

Daksha Yadav[1], **Xiaoli Zhang**[1], **Boyang Tom Jin**[1], **Prakash Krishnan**[1], **Des Clarke**[1]

[1]Amazon
{dakyadav, zhasabri, boyanjin, prakask, clarkede}@amazon.com

Abstract

Timely and accurate reconciliation of the company's financial information is an important internal control over the company's financial reporting to support quarterly and annual external financial compliance activities. However, due to the complexity of typical end-to-end business system integrations, the process to research and investigate reconciliation items can be manual and time consuming. This paper proposes a virtual assistant to enhance the control process by providing end users with tools that will expedite the reconciliation, research and validation process to improve accountants' productivity during financial reconciliation. We employ a large language model (LLM) to enable conversational interactions using natural language. Users can pose queries related to investigating unreconciled transactions, and receive relevant explanations and recommended actions. To map user questions into executable SQL, we use a retrieve-and-refine strategy with retrieval augmented generation (RAG). User questions are encoded into vector embeddings and indexed. Given a new question, relevant examples are retrieved and few-shot prompting constructs an SQL generation prompt for the LLM. The generated SQL is executed to retrieve information from databases. The LLM can invoke additional agents and tools to perform more complex operations over the queried dataset, such as generating and executing an adjustment. We optimize prompt engineering to steer the LLM's behavior, such as providing accounting terminology, instructions, and table schema details. During evaluation, the proposed architecture achieves 95% accuracy in generating correct SQL queries for real-world user questions related to account reconciliation.

Introduction

The financials closing process occurs at the end of each month, where companies (1) identify accounts for closing, (2) record and post the closing entries, and (3) prepare financial reports for quarterly and annual external financial compliance activities (Wild, Shaw, and Chiappetta 2022). During this process, timely and accurate reconciliation of the company's financial information is leveraged as an important internal control, where companies reconcile all accounts that could contain a significant or material misstatement and post all necessary adjustments to the general ledger in a timely manner. As part of this process, reviewers scan the reporting for accuracy indicators and sign off on checklist items, including using data from comparative periods (prior month, prior quarter, prior year), investigating and documenting large-dollar, large-percentage and other anomalies in variances (Kelso 2011).

The reconciliation, research and review process is tedious, as it requires following standard operating procedures (SOPs) to gather data and create/review multiple reports sourced from multiple systems, and then create the end user report with the detail required to execute all month-end deliverables such as journal-entry posting (accruals, recurring-entry setup), financial system querying, financial system report processing, consolidation rules, allocation processing, financial system maintenance and tax rate processing. It is important to accomplish reconciliations in a shortened time frame, so that the company can identify and correct errors before filing their financial statements.

This paper proposes a large language model (LLM)-based virtual assistant to support accounting teams in the financial reconciliation process. The system utilizes a pre-trained LLM to enable natural language interactions with users, allowing them to pose queries and receive relevant responses. The conversational interface is designed to accelerate users' analysis and workflows by providing on-demand access to information and recommended actions related to reconciling accounts. This virtual assistant simplifies the process as it works with the relevant datasets and tools in the backend while presenting a single window interface for users. Powered by LLMs, the virtual assistant is able to process queries and instructions from the users in natural language. The solution is designed to make the LLMs aware of the specific context of the accounts and transactions that are handled in that organization. To create this solution, we supplement the LLM's knowledge using a dataset of common queries, accounting terminology, and reconciliation procedures. The system is optimized to understand user intent from natural language input, and respond with clarifications, explanations, and procedural guidance tailored to the reconciliation context. We evaluate the assistant's capabilities through user studies with accountants, demonstrating its ability to correctly interpret requests, retrieve accurate information, and suggest appropriate next steps to drive the reconciliation process forward. Overall, this research presents a novel

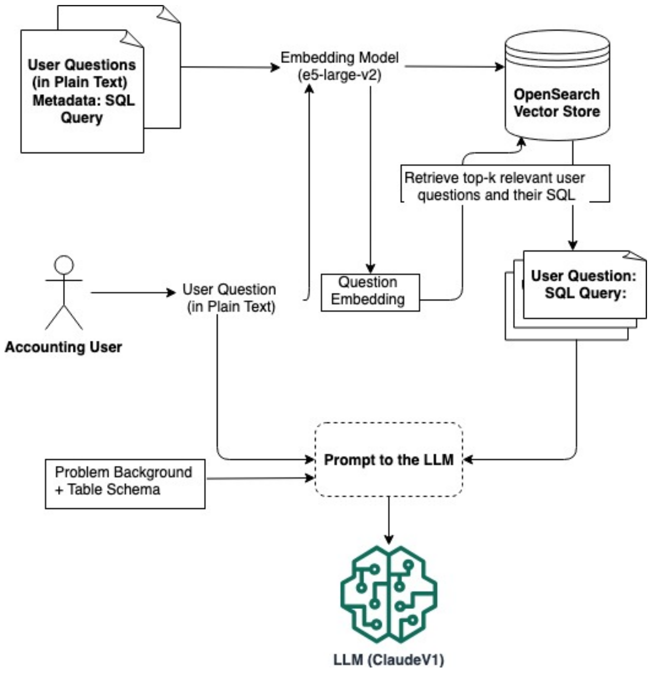
User Questions (in Plain Text) Metadata: SQL Query
Embedding Model (e5-large-v2)
OpenSearch Vector Store
Retrieve top-k relevant user questions and their SQL
User Question: SQL Query:
Accounting User
User Question (in Plain Text)
Question Embedding
Problem Background + Table Schema
Prompt to the LLM
LLM (ClaudeV1)

Figure 1: Illustrating different steps in the framework for the proposed Generative AI based Virtual Assistant for Reconciliation Research.

application of large language model technology to automate aspects of the reconciliation workflow. The conversational assistant aims to enhance accountants' productivity and efficiency when performing this critical finance function.

Approach and Architecture

The proposed virtual assistant is optimized to understand user intent from natural language input, and respond with clarifications, explanations, and procedural guidance tailored to the reconciliation context. One of the critical features of this system is that it should be able to take a natural language question from the user and convert it into a corresponding SQL query that can be executed against a database. This enables the system to understand the user's intent from their question and retrieve the desired information from the database. In this paper, we explain how this was achieved.

To accomplish this, we use a retrieve-and-refine strategy leveraging retrieval augmented generation (RAG) (Liu et al. 2023) as shown in Figure 1. The system first retrieves relevant examples from an embedding index built from a dataset of (question, SQL) pairs. Specifically, the questions are encoded into dense vector representations using a pretrained embedding model (e.g. SentenceTransformers e5-large-v2 (Wang et al. 2022)). These embeddings are indexed in a vector search engine such as OpenSearch (OpenSearch 2023).

During inference, when the user enters a question related to reconciliation process, the system converts the question into a numerical representation using the same embedding model. Using this embedding, we query the vector store to find the top-k most relevant user questions. For retrieving "similar" or "relevant" items, distance-based techniques are typically utilized. At the same time, we also retrieve their corresponding SQL query which is stored as the metadata.

Given the retrieved question-SQL pairs, we create the prompt for the LLM to generate the final SQL query for the input user question using a few-shot prompting technique. With few-shot prompting, we provide the model with a set of high-quality examples, each consisting of both the input and desired output on a target task. As the model first sees examples, hypothetically, it can better understand the human intention and criteria for what kinds of answers are desired. Therefore, few-shot learning often leads to better performance than zero-shot (Gao et al. 2023).

The prompt also contains relevant information about the problem domain and information about the table schema. We add necessary information about match sets and open match sets. We also supply a description about important columns such as the account, company code, and country to help LLM in generating the SQL query.

The LLM-generated SQL statement is post-processed prior execution, such as applying guardrails and error checks. The SQL statement is then executed on the appropriate database and the results are presented to the user.

Embedding Dataset

To understand the customer requirements for the assistant, our team conducted interviews where we asked customers for a set of questions that would likely pose to the assistant, steps to obtain the answer, and desired responses. The primary use cases involved querying for open account balances across different dimensions like company code, country, etc. Examples:

- What is the balance for account 12345?
- What is the balance of 12345 at the company code level?
- Is there a ticket I could reference for issue XYZ?
- What are some possible reasons why transaction ABC is not reconciled?

In order to build a robust question pool, an LLM was then employed to increase the quantity and diversity of user questions, such as by asking the LLM to generate variations on the questions. An example prompt:

Given the following user question, generate a list of similar questions that an accountant may ask. User question: What is the balance for account 12345?

Response by the LLM:

- *What is the ending balance of account 12345 as of today?*
- *What is the balance of account 12345 as of the last day of the month?*
- *What is the average balance of account 12345 over the past few years?*

We also included variables such as country code, company code, and date range to increase the question pool.

Given this set of questions, the next task was to generate corresponding SQL queries and validate the response by running it against the actual database. There are two ways to generate the queries:

1. Manual: A human annotator familiar with SQL can write the corresponding SQL queries. The queries are then validated by running it against the database.
2. LLM-aided generation: An LLM is used to generate SQL queries. A human expert then validates the queries.

The final enriched question-SQL dataset is then used for embedding purposes.

Embedding of Questions

The RAG architecture primarily relies on embedding or encoding items/documents of interest (external to LLM knowledge) in a vector store and then retrieving items "similar" to input user question during inference. Embedding is the process of converting text into a numerical representation that captures the meaning of the text. This is done by training a neural network on a large corpus of text. The neural network learns to represent each word in the corpus as a vector of numbers. This vector captures the semantic and syntactic relationships between the word and other words in the corpus. To perform this conversion from text to the numerical domain, typically off-the-shelf embedding model are used which have been trained on large volumes of existing benchmark datasets. For this paper, we use the intfloat/e5-large-v2 embedding model (Wang et al. 2022). The question-SQL dataset containing 100+ user questions is indexed into the vector store.

Number of Relevant Examples to Retrieve: *k*

When determining the number of similar questions to retrieve for each query, there is a trade-off between providing enough examples for the large language model to generate high-quality responses, while also staying within computational constraints. Specifically, the LLM we are using has a maximum input token limit that restricts the total number of tokens that can be processed for each query. Sending too many question examples risks exceeding this limit and failing to generate a response. At the same time, providing too few examples starves the LLM of the data it needs to understand the contextual similarities between questions and generate relevant responses.

Through experimentation, we found that retrieving 3 similar questions per query struck an effective balance given our particular LLM's input token limit. This allows the model to gather sufficient signal about the semantic and contextual relationships between similar questions, while staying within the computational budget per query. When $k = 1$, we obtain 80% accuracy, as compared to 83% when $k = 2$.

LLM Used

For the purpose of this paper, we use Claude V1-Instant (Anthropic 2023) which is a LLM created by Anthropic. It utilizes a Transformer-based neural network architecture trained on Anthropic's Constitutional AI dataset to generate natural language responses. The model architecture consists of trainable embeddings, a transformer encoder, and an autoregressive decoder. The embeddings convert the input text into vector representations that capture semantic meaning. The transformer encoder then processes these embeddings, learning contextual relationships between words and concepts through multiple self-attention layers. The decoder takes the encoder output and generates a response word-by-word, with each new word conditioned on the previous words and encoder context.

Prompt Engineering

Prompt engineering, also known as in-context prompting, refers to methods for how to communicate with LLM to steer its behavior for desired outcomes without updating the model weights. The effect of prompt engineering methods can vary greatly among models, thus requiring heavy experimentation and heuristics. We started with a basic prompt:

Based on the following example user questions and corresponding SQL statements, your job is to draft a SQL query for the input question.

Examples: {k retrieved examples}

Question: <user question>

Output SQL Query:

For the above prompt, we noticed that the LLM started to hallucinate variable names, account numbers, etc. To solve this issue, we iteratively updated the prompt and evaluated the LLM-generated SQL query. Some of our findings are described below:

- **Reinforce the instructions/task details before and after the examples**. This ensures that the LLM does not forget the instructions. Example:

  Based on the following example user questions and SQL, your job is to draft SQL query for the input question. Do not create new columns. In the SQL query, the account number should be in quotation marks.

  Examples: {k retrieved examples}

  Based on the above user questions and corresponding SQL, your job is to draft SQL query for the input question. Do not create new column names. Do not add date in the SQL query if the user question did not contain any date. Do not create new variables which are not present in the user question.

- **Explicitly provide instructions about how the LLM should generate the final output**. Example:

  Do not create new column names. Do not add date in the SQL query if the user question did not contain any date. Do not create new variables which are not present in the user question. In the SQL query, the account number should be in quotation marks. Generate the SQL query within <sql>and </sql>tags.

- **When injecting the table schema, highlight important column description, their datatype, and typical format**. Example:

  TABLE_AAA only contains the following columns:

  account which is a 5-digit code, company represents 2-character company code, issue id is an integer identifier for the issue, description is the string description of the issue.

- **Provide relevant background about the problem domain**. Example:
  TABLE_AAA contains open match sets. A match set is responsible for grouping two related journal lines. A match set is when the balance nets to zero. A non-zero net balance indicates an issue. The TABLE_AAA tracks these issues. For each open match set, an issue id is assigned. Issue description is documented in column description. An issue can have multiple open match sets assigned to it.

Evaluation

To evaluate the efficacy of the proposed architecture, we constructed an evaluation dataset containing over 80 questions derived from embedding sets and original user inquiries. The dataset excluded complex questions requiring nested queries or combining multiple embedding set questions. We characterized each question based on the presence of GROUP BY, LIMIT, or WHERE clauses to determine SQL generation complexity.

We input the evaluation questions into the proposed LLM pipeline and obtained SQL responses. We executed these SQL responses against a database table containing the dataset. Queries triggering database execution errors were automatically labeled INCORRECT. For successful executions, we manually evaluated the generated query and output, labeling each CORRECT or INCORRECT.

We computed LLM output accuracy as:

$$Accuracy = \frac{Number\ of\ correct\ SQL\ queries}{Total\ number\ of\ queries} \qquad (1)$$

Based on this metric, the proposed architecture achieved 95% accuracy on the evaluation dataset.

We also conducted an ablation study to understand the contribution of the retrieve-and-generate (RAG) component to the overall approach. Specifically, we set the number of retrieved question-SQL pairs $k = 0$ when generating the prompt for the large language model (LLM), meaning that no relevant examples were retrieved or included in the prompt. In this ablation setting without the RAG component, we observed an accuracy of 74% on the SQL generation task. This demonstrates that the RAG component provides a considerable boost in accuracy over using the LLM alone, highlighting the importance of retrieving and incorporating relevant examples into the prompt for the LLM. The RAG component likely improves performance by providing useful in-domain examples that make the SQL generation task more clear and grounded for the LLM.

LLM Agents for Task Automation

While the virtual assistant enables users to query data and receive explanations via natural language, LLMs also have potential to act as agents that can autonomously execute repetitive reconciliation tasks on the user's behalf. The data retrieved from the user's initial questions lays a foundation for the agent to take further action.

One example is automating the creation of adjustments and journal entries. By querying tables that link open match sets to payment statuses, the LLM agent could identify groups of unreconciled transactions and infer potential reasons based on the payment data. For instance, if a payment failed due to decline by the payment processor, the agent may suggest classifying the transaction as a bad debt write-off. The agent could then prompt the user to confirm the write-off, and if approved, automatically generate and submit the adjustment journal entry without the accountant needing to perform the tedious manual steps.

LLM agents could also handle application of foreign exchange rates to query results. Rather than requiring accountants to retrieve rates from an API and manually convert values in Excel, the agent could query the API, retrieve the rate for the desired currency, and apply the conversion to the result set before presentation to the user. By coding these complex multi-step workflows as agent logic, reconciliation tasks can be automated while retaining human oversight.

Conclusion

The adoption of LLMs as virtual assistants in the accounting domain has the potential to enhance productivity for accountants during the critical month-close reconciliation process. By translating natural language questions into executable SQL and natively incorporating tools such as adjustments, the assistant simplifies access to the information needed for reconciliation research, accelerating month-close and freeing up accountants' time for higher judgement tasks.

References

Anthropic. 2023. Claude. https://www.anthropic.com/. Accessed: 2023-11-16.

Gao, D.; Wang, H.; Li, Y.; Sun, X.; Qian, Y.; Ding, B.; and Zhou, J. 2023. Text-to-SQL empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Kelso, K. 2011. Building blocks of a successful financial close process. *Journal of Accountancy*, 212(6): 18.

Liu, J.; Jin, J.; Wang, Z.; Cheng, J.; Dou, Z.; and Wen, J.-R. 2023. RETA-LLM: A Retrieval-Augmented Large Language Model Toolkit. *arXiv preprint arXiv:2306.05212*.

OpenSearch. 2023. OpenSearch. https://opensearch.org. Accessed: 2023-11-16.

Wang, L.; Yang, N.; Huang, X.; Jiao, B.; Yang, L.; Jiang, D.; Majumder, R.; and Wei, F. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.

Wild, J.; Shaw, K.; and Chiappetta, B. 2022. *Financial and Managerial Accounting*. McGraw Hill.