Journal of Software Engineering and Applications, 2025, 18(1), 44-65
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Intelligent ETL for Enterprise Software Applications Using Unstructured Data

Manthan Joshi, Vijay K. Madisetti

School of Cybersecurity and Privacy, Georgia Institute of Technology, Atlanta, GA, USA
Email: manthanjsh@gmail.com, vkm@gatech.edu

How to cite this paper: Joshi, M. and Madisetti, V.K. (2025) Intelligent ETL for Enterprise Software Applications Using Unstructured Data. *Journal of Software Engineering and Applications*, **18**, 44-65.
https://doi.org/10.4236/jsea.2025.181003

Received: December 27, 2024
Accepted: January 27, 2025
Published: January 30, 2025

Abstract

Enterprise applications utilize relational databases and structured business processes, requiring slow and expensive conversion of inputs and outputs, from business documents such as invoices, purchase orders, and receipts, into known templates and schemas before processing. We propose a new LLM Agent-based intelligent data extraction, transformation, and load (IntelligentETL) pipeline that not only ingests PDFs and detects inputs within it but also addresses the extraction of structured and unstructured data by developing tools that most efficiently and securely deal with respective data types. We study the efficiency of our proposed pipeline and compare it with enterprise solutions that also utilize LLMs. We establish the supremacy in timely and accurate data extraction and transformation capabilities of our approach for analyzing the data from varied sources based on nested and/or interlinked input constraints.

Keywords

Structured Data, Relational Model, LLM-Powered Agents, Field-Level Extraction, Knowledge Graph

1. Introduction

Organizations use both structured and unstructured data, often organized in tables, as part of extract, transform, and load (ETL) processes prior to ingestion into business process software. Structured datasets, like relational databases, can be analyzed using SQL. However, crucial business information is often stored in insecure PDFs, such as invoices, contracts, and manuals. Analyzing data structures from these formats poses challenges, hindering precise calculations and insights, while compromising privacy and security.

We shall now examine a prevalent use case of significance to enterprises,

pertaining to supply chain procurement processes. This case study will be elaborated upon to demonstrate the application of our proposed methodology.

1.1. Procurement Workflows in Enterprises

Figure 1 depicts a basic manual procurement workflow involving a Vendor, Manager, Purchase Agent, Recovery Agent, and Payee, each conducting their roles sequentially using traditional communication methods like emails, ticketing, document submissions, phone, or in-person talks. This dynamic process often requires repeated confirmations and verifications, making human-based systems prone to errors [1].

The manual procurement process starts with vendors standardizing documents, such as text-based contracts, clauses, and structured quotes, usually in PDF format. These documents outline product pricing and terms and go through a review and approval process to meet the organization's objectives.

The manager, after reviewing customer requests, instructs the purchasing agent to start procurement. The agent communicates with the vendor through the manager to negotiate and adjust quotes based on quantity. This involves relaying the user's requirements to the vendor. The recovery agent oversees delivery tracking, verifies items, and resolves disputes via the manager. Once quotes are final and deliveries confirmed, the payee processes payments. Direct vendor communication to confirm quotes is sometimes needed, as shown by a dotted line in Figure 1.

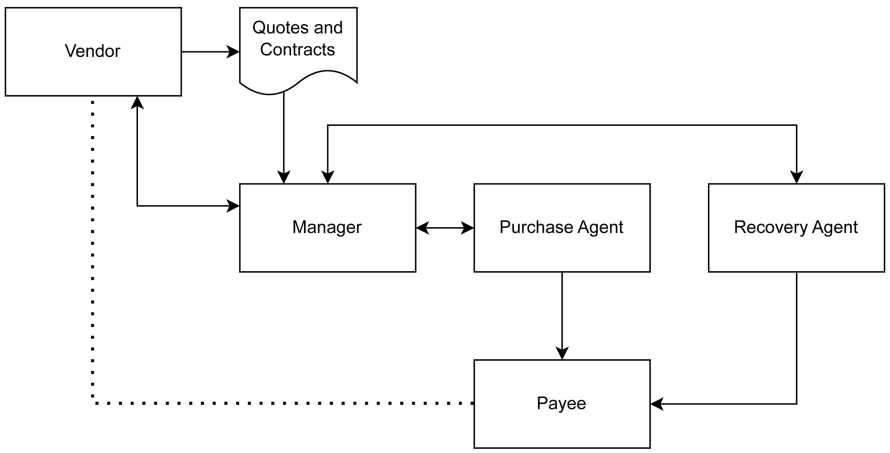

Figure 1. Typical manual procurement process.

To reduce errors stemming from human involvement in communication channels, both between users and vendors and within an organization, transitioning from manual processes to an AI-based pipeline is now a research focus. The LLM-based communication framework promises reduced time consumption, as well as more direct and efficient communication [2].

Centralized data processing and extraction are enhanced by language model integration, streamlining documentation-heavy tasks into digital processes. Instantaneous complex filtering and data retrieval, from both structured and

unstructured data, rely on query speed instead of human action. Preserving memory and user context reduces workflow redundancies, boosting collaboration efficiency, reducing bottlenecks and errors, and offering flexible document handling. Recently developed Large Language Models (LLMs) promise to significantly advance ETL processes beyond manual limitations.

1.2. Enterprise Workflows and Data Formats

Relational databases effectively store and analyze structured data, but struggle with formats like PDFs and text files due to missing processing markers. Tools using OCR or LLMs to extract text from PDFs fall short of the accuracy of database queries and often convert data to JSON, which can introduce errors [3]. While LLM improvements reduce inaccuracies, they still present risks in critical areas like hospitals. Workflows for structured and unstructured data are often kept separate to preserve solution versatility, leading to fragmented insights and inefficiencies. Consequently, organizations fail to fully utilize their data, particularly when vital information is scattered across various sources. In finance, legal, and industry sectors, analyzing relational data is vital. Not being able to examine structured data with unstructured documents can be harmful. For instance, auditors might need to link PDF invoices to SQL transaction records. Lacking a unified framework leads to manual processing, raising error rates. As reliance on both data types grows, a cohesive system is essential.

Contemporary AI-driven tools leveraging Large Language Models (LLMs) for prompt analysis and JSON format outputting may obstruct subsequent data processing required for comprehensive analysis. Such tools handle extractions as isolated tasks, necessitating manual transformation for continued utilization. Furthermore, the LLM is often compelled to reprocess each prompt, notwithstanding its potential relevance to previously supplied context.

2. Problem Statement

The identified issues in the ETL solutions that exist today are:

- Inconsistent formats across datasets, leading to fragmented workflows.
- Limited querying capabilities for unstructured data, impeding advanced analytics.
- Manual interventions are required to merge and analyze data across different sources.

Addressing these challenges, especially for structured data saved in formats such as PDFs, requires an approach that utilizes LLM along with the SQL querying and RAG generating capabilities to make sense of the data and also query as necessary [4]. Leveraging both LLM-RAG and SQL agents makes processes more specific to operations. LLMs, trained on vast datasets, excel in processing prompts and generating outputs, interpreting user requests, assigning tasks to specialized agents, and creating queries for structured data. SQL excels in retrieving structured data from relational databases, while RAG efficiently retrieves data from

unstructured documents and formulates responses. Using SQL and RAG agents based on data type, user prompts, and storage location enhances performance.

This paper introduces a novel method for extracting both structured and unstructured data and performing a web search for the entered prompt to generate the most relevant response. The project structure is shown in Figure 2. The pipeline employs three LLMs; a primary LLM analyzes prompts alongside a knowledge graph for context. Details of the SQL agent, RAG agent, and Tavily web search agent are provided in later sections.

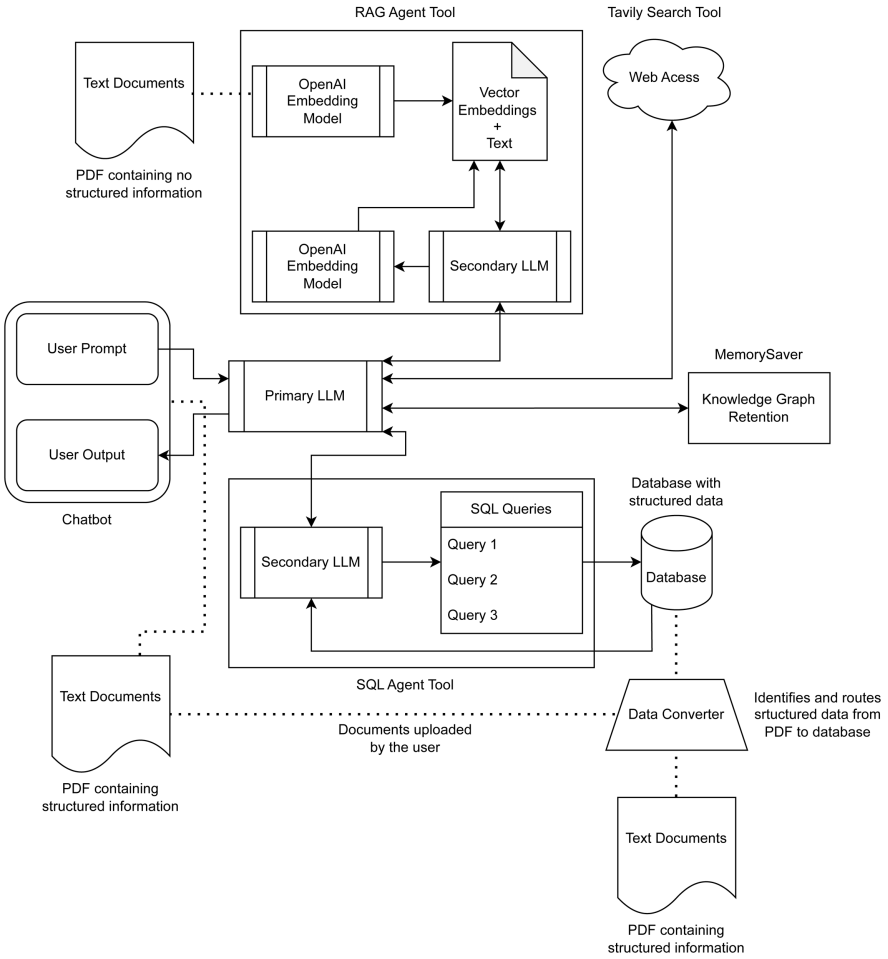

Figure 2. Project structure block diagram.

3. Related Work

The use of LLMs for extracting structured and unstructured data is a new and rapidly growing field [5]. There are few existing contributions [6] that address certain entrepreneurial analytical requirements today. Tools have been created to turn unstructured PDF data into structured formats for final output, but these outputs cannot be modified using prompt history, query state, and context [7].

Key existing enterprise solutions include:

- Unstract: A tool that extracts fields from PDFs, emails, and text, generating

JSON outputs.

- Tabula: Open-source tool to extract tables from PDFs and export to CSV or JSON.

Tools like unstract and Tabula are effective for raw data extraction but are limited for advanced analytics as they do not integrate with relational databases [8] [9].

3.1. Limitations of Existing Tools

3.1.1. Lack of Relational Data Handling

These existing instruments facilitate the extraction and conversion of raw data into JSON or CSV formats; however, they do not retain data in SQL-compatible formats. This limitation hinders their capability to support relational queries and joins with other structured data sources. Furthermore, unstract solely accepts data in PDF format.

3.1.2. Limited Analytical Capabilities

The current tools are primarily designed for field-level extraction; however, they lack the capability to directly facilitate advanced relational queries, including conditional joins or aggregations across various tabular fields derived from multiple documents within the database [10]. This limitation further constrains the potential for bidirectional interaction with the database. JSON-based outputs necessitate additional transformations to integrate effectively with SQL databases, thereby introducing further complexity.

3.1.3. Limited Use of AI for Contextual Analysis

Moreover, the majority of extant solutions depend on predefined extraction models, which lack the capacity for context-sensitive processing [11] [12].

3.2. Solution Requirements

Organizations require a comprehensive pipeline that can:

1) Extract and store structured PDF data as relational data to facilitate SQL-based analytics.

2) Seamlessly merge and query structured and unstructured data without needing additional ETL processes.

3) Validate and standardize extracted data using AI-powered models to ensure accuracy and usability.

3.3. Our Contribution to the Existing Solutions

We now identify the main features of our solution, Intelligent ETL:

3.3.1. Structured Content Extraction and Transformation

We developed a Python-based workflow specifically designed for the detection and extraction of structured tabular data from PDF documents. This facilitates the seamless integration of extracted PDF data into pre-existing SQL databases,

thereby enabling the execution of relational operations and complex queries. Additionally, we incorporate procedures for the extraction of unstructured data within our pipeline.

3.3.2. GPT-Powered Validation and Field Identification

By incorporating large language models (LLMs) such as GPT-3.5-turbo, our method facilitates dynamic detection and verification of fields, thereby ensuring precise identification of essential fields. This approach affords greater flexibility and adaptability compared to predefined extraction models. Additionally, we utilize the primary LLM for output validation by scrutinizing it against the input prompt and activating the suitable tool when necessary. LLMs are observed to be the best tool for taking context-aware computations into account [13]-[16].

3.3.3. ETL Pipeline Unification

We achieve a convergence of the ETL pipeline that processes both structured SQL data and unstructured PDFs, integrating them within a unified relational data model for analytical purposes. This integration represents a novel advancement, as traditionally, unstructured and structured datasets necessitate distinct pipeline processes, despite the possibility of PDFs harboring inherently structured data. In contrast to tools like unstract, our approach preserves the relational aspects of PDF data by embedding them within SQL tables. This methodology facilitates the direct interrogation of data using SQL, thus eliminating the requirement for additional transformations. This methodology reduces the requirement for separate ETL processes. Furthermore, it enhances the capacity for sophisticated analytics by enabling the execution of complex SQL queries on both structured and unstructured data, while maintaining data integrity through the normalization of extracted data into relational formats. The proposed solution is designed to be scalable, accommodating large datasets, and adaptable for implementation across various projects. Additionally, it supports the storage of xlsx, csv, pdf, and txt formats within databases, thus enabling the Intelligent ETL pipeline to conduct analyses on the encompassed data. In contrast to the unstract approach, this pipeline stores SQL data as opposed to JSON formatted data, thereby facilitating advanced, context-aware querying.

Through the integration of structured and unstructured data within a unified relational model within IntelligentETL, organizations can be empowered to conduct sophisticated analytics and reporting with ease. The implementation of GPT-based AI models introduces an additional layer of intelligence, rendering the system flexible and responsive to diverse user queries. We develop a chatbot-based procurement application utilizing Gradio and LangChain technologies and subsequently evaluate our assertions with respect to “unstrict”.

The reliance on LLMs especially under high-load scenarios, poses a possibility of encountering computational challenges such as computational cost and performance efficiency. This is because LLMs are resource-intensive and may struggle to maintain minimum latency while dealing with large datasets or while processing

concurrent queries.

The mentioned challenges are mitigated in the proposed pipeline as the architecture incorporates a modular design that enables task-specific agents such as SQL and RAG and utilizes capabilities of the knowledge graph to handle discrete operations without fully engaging all the LLMs for every step of the process and for every query in the subsequent chat progression. The knowledge graph works as an efficient caching mechanism thus enabling the pipeline to produce accurate answers for context-relevant queries. Additionally, it also nullifies the requirement of even activating respective LLM-powered agents in the case of context-ready or repetitive queries, thus enabling the pipeline to be scalable.

This enhancement of data generation capabilities, coupled with the ongoing advancements in data storage, suggests an increasing importance of LLM-directed query agents and user-database interactions structured around meticulously designed knowledge graphs in the foreseeable future. Knowledge graphs elucidate the trajectory of system control flow, thereby guiding both developers and users in the creation of applications that encourage question formulation in a manner that, when processed through advanced ETL systems, is likely to yield precise cost-effective outcomes [17]-[19].

Distilled versions of GPT could be further researched for further enhancement in the direction of making the pipeline ready for enterprise-scale deployment.

4. Methodology

4.1. Unstructured Content Extraction and Transformation

Algorithm 1 PDF-to-SQL and Database Merging Pipeline

PDF file path: *pdf_path*, Source DB path: *src_db*, Destination DB path: *dest_db*
Extracted data stored in a SQLite database and merged into the destination database
Function extract_text_from_pdf(*pdf_path*): Extracts and returns the plain text from all pages of the PDF.
Function extract_tables_from_pdf (*pdf_path*): Extracts all tables from the PDF and returns them as a list of DataFrames.
Function save_to_sqlite (*dataframes*, *db_path*): Saves each DataFrame in the list to the SQLite database at *db_path*.
Function merge_databases (*src_db*, *dest_db*):
for each table in *src_db* **do**
  if table exists in *dest_db* **then**
    Skip merging (to avoid duplicates).
  else
    Copy table structure and data to *dest_db*.
  end if
end for
Main Pipeline Execution:
Run pdf_to_sql_pipeline (*pdf_path*, *src_db*) to extract
PDF content and save it to SQLite.
Run merge_databases (*src_db*, *dest_db*) to transfer tables to the destination database.

An in-detail workflow of an automated layout that detects tabular data in PDFs, extracts and transfers it to an SQLite database(s), and transfers it to the destination database that is specified, is shown in Algorithm 1. The solution leverages time-tested Python libraries to handle structured and unstructured data efficiently. pdfplumber is used to extract plain text from multiple PDF sources, camelot is used to extract detected tables from PDFs, pandas is used to refine tabular data and bring it into a structure, and sqlite3 and create _engine from sqlalchemy is used to connect the source and the destination of the data transfer and to save each data-frame in the database respectively. The pipeline is not only designed for scalable data ingestion but also for a versatile merger of multiple databases. Here an end-to-end PDF data merging to a predefined database is achieved. The Data Converter from **Figure 2** depicts this workflow. Enterprise solutions exist that carry out similar tasks. This ETL pipeline is flexible enough to be used with those solutions as well. However, this simple workflow achieves accurate results with comparatively less to none error expectancy. Analytics of structured data presented in PDF is then ready to be carried out based on user prompts, the methodology will be explained in subsequent sections of this paper.

To mathematically represent the data matching and database merging operations for two data sets in different formats, let us consider the following model:

First, the tables are identified by the module and extracted from PDFs using efficient Python-AI libraries. Then:

4.1.1. Saving Extracted Data Frames to SQLite Database

For a given set of DataFrames, D, each DataFrame is saved in an SQLite database:

$$D = \{df_1, df_2, \cdots, df_m\} \tag{1}$$

$$\forall df_i \in D, \quad \text{SQL}_i = \text{Save}(df_i) \tag{2}$$

where:

- D is the set of extracted Data-frames.
- df_i is the i^{th} Data-frame in the set.
- $\text{Save}(df_i)$ saves df_i to the SQLite database.

4.1.2. Database Merging Using Set Theory

Set theory is used to model the merger of source and destination databases:

$$M = S \setminus D \tag{3}$$

$$\forall t \in M, \quad D = D \cup \{t\} \tag{4}$$

where:

- M is the set of tables in the source database not present in the destination.
- S is the set of all tables in the source database.
- D is the set of tables in the destination database.
- t is a table in the set M.

4.1.3. Text Extraction from PDF

Finally, the extracted text from each page is concatenated:

$$\text{Text} = \bigoplus_{i=1}^{n} T_i \tag{5}$$

where:

- T_i is the text extracted from the i^{th} page.
- $\oplus$ denotes string concatenation.

4.2. Prompt Inspection and Agent Identifying Block

The algorithm block of prompt inspection and agent identification is an intermediate block in the system and the next step after the structured and unstructured content ingestion. Three LangChain-linked LLM-powered tools have been designed for the system to function as intended. These are respectively, the SQL agent tool block, the RAG-based retrieval module tool block, and a Tavily-based web search tool block. Each block has its own LLM agents for content retrieval. However, this block employs a GPT-4o-mini primary agent LLM to dynamically study the prompt's intent as well as content and then identify the tool that is most probable for handling the database with respect to the user's input prompt and forwards the user prompt to the respective agent block. It also takes into account the previous query history based on the knowledge graph for context understanding. Additionally, a feature that allows users to upload documents containing structured information adds flexibility for user interaction. The structured data is extracted and stored in the database. The LLM agent ensures that the extracted data aligns precisely with both the entered prompt and the associated data from the PDF document uploaded by the user. The primary LLM agent's logic for prompt inspection is enclosed within the route_tools() method as seen in Figure 3. This method checks whether the question is associated with:

- The database query which requires SQL operations,
- The sequential sentence extraction based on RAG,
- The general, human-like conversation, or
- The web search leveraging the Tavily search.

```
def route_tools(
    state: State,
) -> Literal["tools", "__end__"]:
    """
    Inspects the latest message in the state to determine if a tool needs to be invoked.
    If tool calls are detected, it routes to the 'tools' node; otherwise, it ends.
    """
    if isinstance(state, list):
        ai_message = state[-1]
    elif messages := state.get("messages", []):
        ai_message = messages[-1]
    else:
        raise ValueError(
            f"No messages found in input state to tool_edge: {state}")
    if hasattr(ai_message, "tool_calls") and len(ai_message.tool_calls) > 0:
        return "tools"
    return "__end__"
```

Figure 3. Prompt inspection and deploying tools.

Each agent block is correlated with a specific data type as determined by the user's prompt. For instance, operations conducted at the field level within a relational model—such as calculating the aggregate sum within a table or identifying the minimum and maximum values across all tables in a database—necessitate the engagement of the secondary agent from the SQL agent module to access structured data. Such operations are unattainable using solely a Large Language Model (LLM) agent that employs Retrieval-Augmented Generation (RAG) or functions independently of any auxiliary tool. In contrast, a user prompt necessitating a sequential response through policy search will activate the secondary LLM within the RAG tool module to generate an appropriate response by scrutinizing PDF documents. Moreover, the primary LLM ensures the adequacy of the extracted data in addressing the query before transmitting the final output to the user.

4.3. SQL Agent Tool

The core code for the SQL agent tool is as seen in Figure 4. This agent block handles database-related queries by accessing the relational data stored in the database. As seen in the code, the SQLAgent is implemented to produce queries. The agent, a distinct tool, converts prompts into SQL queries with the guidance of a GPT-3.5-turbo LLM, which interprets the input from the primary LLM. Tests show GPT-3.5-turbo excels in writing and performing database queries compared to GPT-4o-mini. As seen in Figure 5, the mentioned LLM is also responsible for analyzing the retrieved data that is returned in a structured format, against the input prompt and the uploaded document(s), and refactoring the SQL agent to design a more appropriate SQL query if needed. This ensures accurate output aligned with the user's question and related uploaded documents in the database.

```python
class SQLAgent:
    """
    SQL agent that handles queries related to the connected database.
    Translates natural language prompts into SQL queries using an LLM.
    """

    def __init__(self, sqldb_directory: str, llm: str, llm_temperature: float):
        # Initialize the SQL database and the LLM
        self.sql_agent_llm = ChatOpenAI(model=llm, temperature=llm_temperature)
        self.db = SQLDatabase.from_uri(f"sqlite:///{sqldb_directory}")

        # Chain: Map user prompts to table names and execute SQL queries
        category_chain = create_extraction_chain_pydantic(
            Table, self.sql_agent_llm, system_message="Identify relevant tables for the query."
        )
        query_chain = create_sql_query_chain(self.sql_agent_llm, self.db)

        # Combine extraction and query logic into a single pipeline
        self.full_chain = category_chain | query_chain

@tool
def query_sqldb(query: str) -> str:
    """
    Executes a user-defined query on the SQL database.
    Returns query results or raises an error if execution fails.
    """
    agent = SQLAgent(
        sqldb_directory=TOOLS_CFG.sqldb_directory,
        llm=TOOLS_CFG.sqlagent_llm,
        llm_temperature=TOOLS_CFG.sqlagent_llm_temperature
    )
    return agent.full_chain.invoke({"question": query})
```

Figure 4. SQL agent tool.

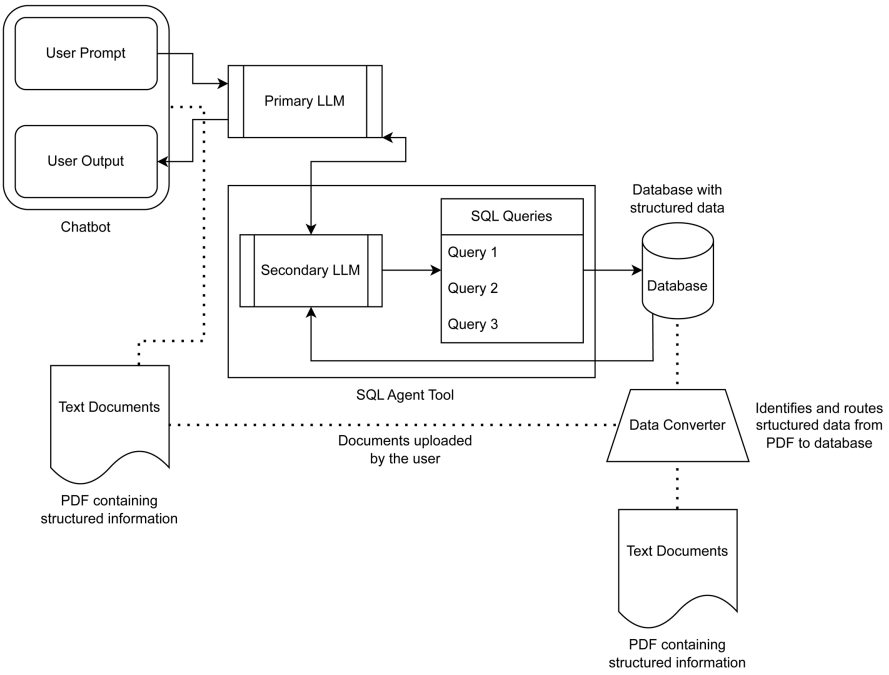

Figure 5. SQL agent.

The SQL agent module performs operations like joins, min/max calculations, and aggregations. For instance, it joins invoice and product tables to calculate total sales per product, which enterprise solutions like Unstract cannot achieve, as discussed in the paper.

4.4. RAG Agent Block

The core code for the RAG agent tool is as seen in Figure 6. The RAG agent module utilizes a Chroma-vector-based search mechanism for the extraction of pertinent unstructured data from diverse sources. Its efficacy is particularly pronounced in the context of sequential text extraction, necessitated by the nature of text vector embeddings and the corresponding text, which is conventionally documented in JSON format. This agent is developed as an independent tool, tasked with the processing of text-based queries via the implemented pdfRAGTool. When the GPT-4o-mini agent identifies PDF-based policy data extraction needs, it triggers the tool in LangChain.

The RAG agent block is as seen in Figure 7. The agent module embeds the document and prompt into vectors using OpenAI's text-embedding-3-small model and stores them in JSON. It queries Chroma-based data for prompt-related information, which is then used by GPT-4o-mini to create a clear user response.

When a prompt involves an uploaded document with RAG models and LangChain, the tool uses vector embeddings to find and extract the most relevant sequence with GPT-4o-mini. The output is then summarized in human-readable format by the LLM within the RAG agent module. RAG is effective at synthesizing information from unstructured data, even in PDFs. GPT-4o-mini is used as both

the primary and secondary LLM for its efficiency, lightweight nature, and free availability.

```python
class pdfRAGTool:
    """
    A tool for retrieving relevant information from PDFs using RAG (Retrieval-Augmented Generation).
    Utilizes vector embeddings and a Chroma vector database for similarity search.
    """

    def __init__(self, embedding_model: str, vectordb_dir: str, k: int, collection_name: str):
        # Initialize the vector database with the specified embedding model
        self.vectordb = Chroma(
            collection_name=collection_name,
            persist_directory=vectordb_dir,
            embedding_function=OpenAIEmbeddings(model=embedding_model)
        )

@tool
def lookup_pdf(query: str) -> str:
    """
    Performs a similarity search on PDFs stored in the vector database.
    Returns the most relevant sections as text.
    """
    rag_tool = pdfRAGTool(
        embedding_model=TOOLS_CFG.pdf_rag_embedding_model,
        vectordb_dir=TOOLS_CFG.pdf_rag_vectordb_directory,
        k=TOOLS_CFG.pdf_rag_k,
        collection_name=TOOLS_CFG.pdf_rag_collection_name
    )
    docs = rag_tool.vectordb.similarity_search(query, k=rag_tool.k)
    return "\n\n".join([doc.page_content for doc in docs])
```

Figure 6. RAG agent tool.

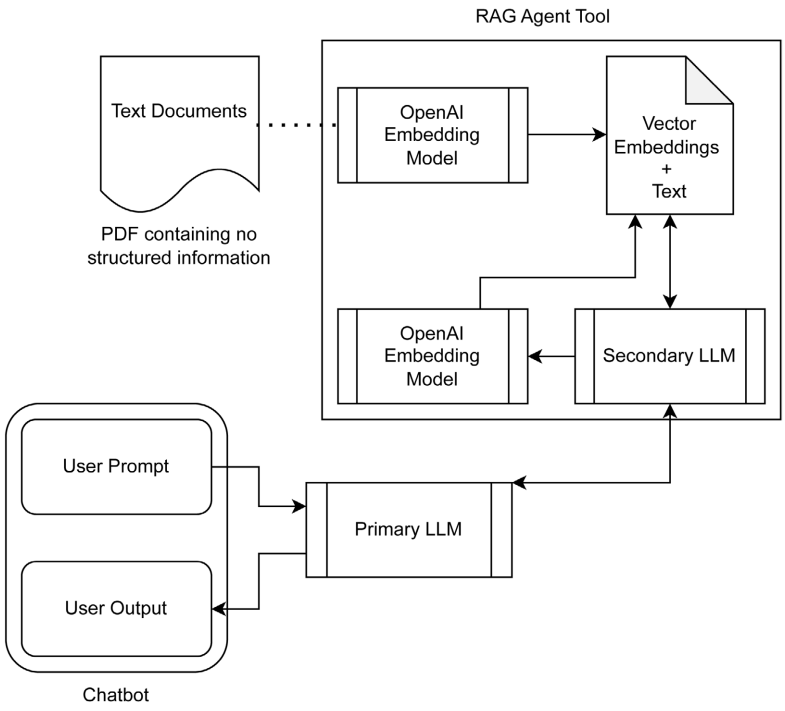

Figure 7. RAG agent.

4.5. Tavily Tool Integration for Web-Based Searches

The workflow also integrates the Tavily tool which is most efficient for web-based searching capability. Tavily is an exclusive web search tool, designed to be easily

integrated into AI-powered applications. If the primary agent deduces that the entered prompt requires web-based search, this tool is invoked. The Tavily tool in this workflow is initiated to return the 3 most relevant website links that most match the user prompt based on the analysis conducted by the primary agent.

4.6. Knowledge Graph Generation

The core code for control flow as depicted by the knowledge graph is as seen in Figure 8. A knowledge graph effectively represents the control flow of ensemble systems, offering a concise view of the workflow and modules while aiding in analyzing interactions for context-accurate queries. This is crucial for designing and optimizing AI applications focused on timely accuracy. All the designed agents along with their respective tools and the user prompts are represented as nodes in a knowledge graph. The control transition between these nodes as seen in Figure 8, is modeled as edges that denote the control flow of the system. The knowledge graph is visualized through high-level control flow modules as seen in the block diagram of the generated knowledge graph in Figure 9. The State graph or knowledge graph is generated by the primary LLM denoting the control flow from the user prompts and the tools invoked, to the output prompt in the chatbot. The memory is saved for the previous state retention and routing in the MemorySaver object from langgraph.checkpoint module by logging checkpoints of the previous states of interaction.

With each interaction by the user, the knowledge graph undergoes dynamic expansion. The StateGraph object is configured and constructed by the principal LLM according to the flow of control across various tools and states. Performance

```
# Load configuration settings for tools and agents
TOOLS_CFG = LoadToolsConfig()
def build_graph():
    """
    Constructs the StateGraph to manage agent invocations.
    This graph integrates a primary LLM with several predefined tools to process user queries and guide interactions dynamically.
    """
    # Initialize the primary LLM with tool-binding capability
    primary_llm = ChatOpenAI(model=TOOLS_CFG.primary_agent_llm,
                             temperature=TOOLS_CFG.primary_agent_llm_temperature)
    graph_builder = StateGraph(State)
    # Load search tool and other external tools
    search_tool = load_tavily_search_tool(TOOLS_CFG.tavily_search_max_results)
    tools = [search_tool, lookup_pdf, query_sqldb]
    # Bind the LLM to the list of tools
    primary_llm_with_tools = primary_llm.bind_tools(tools)

    def chatbot(state: State):
        """
        Executes the LLM with tools bound and generates the next response.
        """
        return {"messages": [primary_llm_with_tools.invoke(state["messages"])]}
    # Define nodes in the graph
    graph_builder.add_node("chatbot", chatbot)  # Chatbot Node
    tool_node = BasicToolNode(tools=[search_tool, lookup_pdf, query_sqldb])
    graph_builder.add_node("tools", tool_node)  # Tool Node
    # Define conditional routing between nodes
    graph_builder.add_conditional_edges(
        "chatbot", route_tools, {"tools": "tools", "__end__": "__end__"}
    )
    # Define transitions between nodes to ensure smooth interaction
    graph_builder.add_edge("tools", "chatbot")
    graph_builder.add_edge(START, "chatbot")
    # Save state checkpoints during execution
    memory = MemorySaver()
    graph = graph_builder.compile(checkpointer=memory)
    # Optionally plot the graph schema for visualization
    plot_agent_schema(graph)
    return graph
```

Figure 8. Pipeline flow.

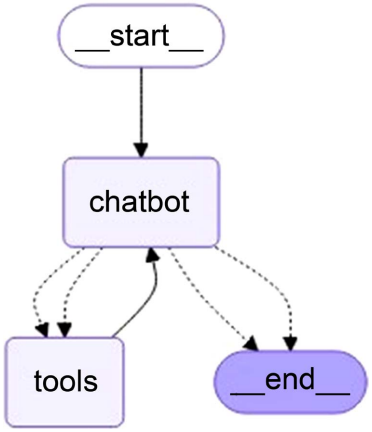

Figure 9. Generated knowledge graph.

metrics are available for examination on the LangChain dashboard. The sequence of inputs, the agents invoked consequently, and the output delivered can thus be scrutinized. The memory preservation mechanism is activated to enable the system to record prior interactions, thereby enhancing the content-awareness and adaptability of ensuing dialogues. This process is referred to as state maintenance within the framework of a knowledge graph representation.

4.7. Integration with Enterprise Software Environments

The modular design of the system, is suitable for integrating the pipeline with enterprise-grade software ecosystems such as Enterprise Resource Planning (ERP) via standardized APIs.

The pipeline as a module is also suitable to be integrated with the sub-modules of ERP such as Customer Relationship Management (CRM) for automating individual parts of the ecosystem.

The pipeline integrates into the ERP as seen in Figure 10. The results for these functionalities are discussed in the subsequent section of the paper. The modules that the IntelligentETL pipeline replaces are:

1) Supply Chain Management (SCM) and CRM: The pipeline is responsible for extracting, validating, and integrating vendor contracts, invoices, purchase orders, and customer requirements into SQL-compatible formats. It enables analytics such as customer order history, and customer behavior insights based on the knowledge graph formation for each customer interaction.

2) Data Warehousing: The pipeline efficiently ingests and centralizes structured and unstructured data into a relational model. This facilitates not just efficient querying but also analytics-ready storage and dynamic data retrieval for enterprise-wide reporting.

3) Financial Resource Management (FRM): The extraction and validation of invoices, customer requirements, and payment data support advanced financial reconciliations through robust and accurate SQL query formulation.

The modules where the IntelligentETL pipeline lacks direct applicability are:

Human Resource Planning (HRP), and Manufacturing Resource Planning (MRP). The pipeline supports inventory and supply chain-related data but does not cater to functionalities specific to MRP and manufacturing processes such as production scheduling, and shop control. Also, it does not address HRP-specific functionalities such as employee records, recruitment processes, and payroll analytics in its current architecture. Designing additional modules or integrations tailored to HRP and MRP use cases is necessary to extend the pipeline's applicability to enterprise applications dealing with the automation of these processes. An additional module specifically designed for resource planning is then necessary to be integrated with the developed pipeline to account for HRP and MRP as per respective application requirements based on the environments such as financial auditing, supply chain management, and legal documentation workflows.

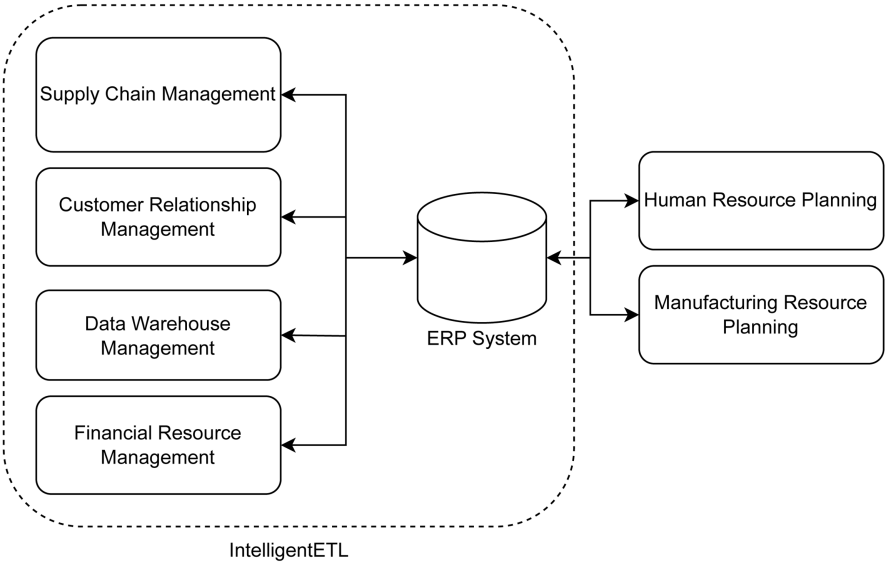

Figure 10. Integration with enterprise resource planning modules.

5. Results and Discussion

5.1. PDF-to-SQL and Database Merging Pipeline

The validation process for data conversion speed is conducted herein. Subsequent to this, the accuracy of structured data extraction will be evaluated, and the resulting metrics will be compared with those derived from unstract.

The time taken for each of the operations is as given:

$$T_{\text{operation}} = t_{\text{end}} - t_{\text{start}} \tag{6}$$

where:

- $T_{\text{operation}}$ is the time taken for the operation to complete.
- t_{start} is the timestamp at the start of the operation.
- t_{end} is the timestamp at the end of the operation.

The total time for the complete transformation workflow is given as:

$$T_{\text{total}} = T_{\text{extraction}} + T_{\text{saving}} + T_{\text{merging}} \tag{7}$$

where:

- $T_{\text{extraction}}$ is the time for extracting tables along with the corresponding text.
- T_{saving} is the time for saving tables to the SQLite database.
- T_{merging} is the time for merging databases.

As seen in Figure 11, the time analysis result for all the operations for data transformation explained above sums up to 0.07 seconds for an A4-sized PDF. This is on par with the enterprise-level software that performs the task of only data conversion.

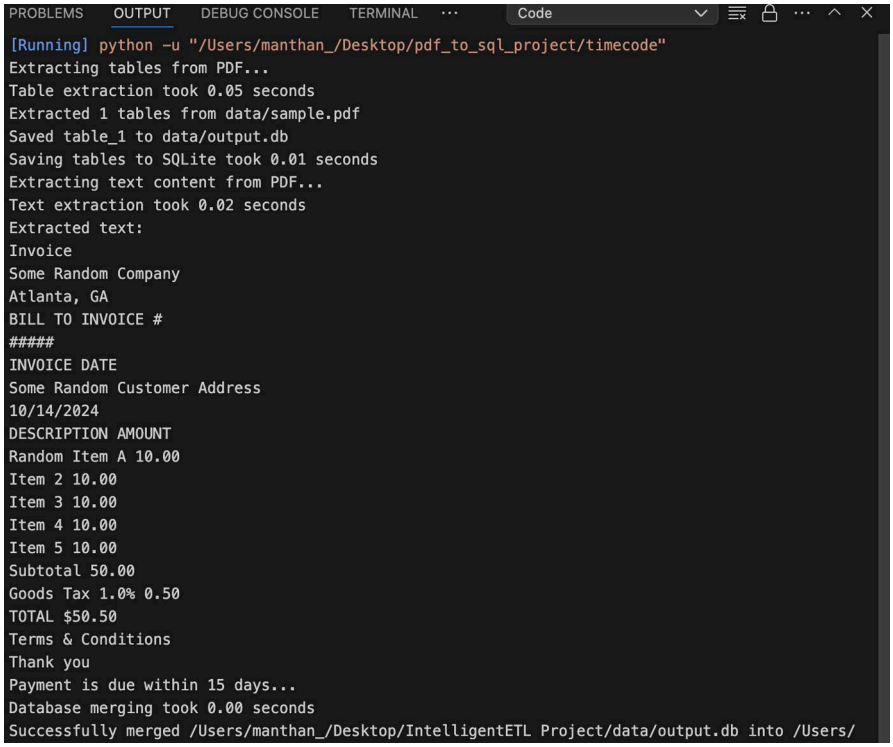

Figure 11. Data transformation time analysis.

5.2. Accuracy of LLM-SQL Based Data Extraction

Figure 12 shows the performance of the LLM-backed SQL agent. The application was structured using LangChain. As such, LangChain offers excellent performance metrics reporting services.

The performance metrics are displayed as seen in Figure 13. The metrics were obtained for the SQL agent deployed via the LLM to facilitate database querying. Upon conducting tests with a variety of prompts, it was noted that the performance of the LLM-based RAG and Tavily agents was comparable to that of the LLM-based SQL agent. Moreover, the performance metrics for any future tools that might be integrated into this pipeline through the LLM could be similarly analyzed. The adaptability to incorporate additional tools seamlessly, coupled

with the ease of performance assessment via LangChain capabilities, significantly enhances the pipeline's customizability—an aspect that is notably absent in the unstract framework.

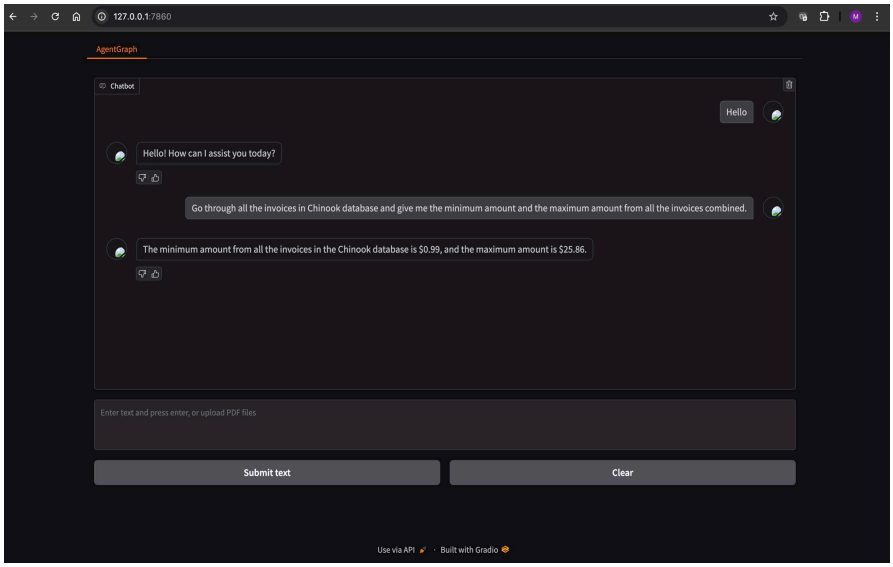

Figure 12. Custom-built chatbot with prompt output for LLM deployed SQL agent.

```python
class SQLAgent:
    """
    SQL agent that handles queries related to the connected database.
    Translates natural language prompts into SQL queries using an LLM.
    """

    def __init__(self, sqldb_directory: str, llm: str, llm_temperature: float):
        # Initialize the SQL database and the LLM
        self.sql_agent_llm = ChatOpenAI(model=llm, temperature=llm_temperature)
        self.db = SQLDatabase.from_uri(f"sqlite:///{sqldb_directory}")

        # Chain: Map user prompts to table names and execute SQL queries
        category_chain = create_extraction_chain_pydantic(
            Table, self.sql_agent_llm, system_message="Identify relevant tables for the query."
        )
        query_chain = create_sql_query_chain(self.sql_agent_llm, self.db)

        # Combine extraction and query logic into a single pipeline
        self.full_chain = category_chain | query_chain

@tool
def query_sqldb(query: str) -> str:
    """
    Executes a user-defined query on the SQL database.
    Returns query results or raises an error if execution fails.
    """
    agent = SQLAgent(
        sqldb_directory=TOOLS_CFG.sqldb_directory,
        llm=TOOLS_CFG.sqlagent_llm,
        llm_temperature=TOOLS_CFG.sqlagent_llm_temperature
    )
    return agent.full_chain.invoke({"question": query})
```

Figure 13. Performance metrics of LLM deployed SQL agent as captured by LangChain.

The query was initiated to determine the minimum and maximum amounts across all invoices within the database. The agent efficiently parsed through the extensive database, providing a 100% accurate response in only 6.50 seconds,

based on 1399 generated tokens. Including the duration required for extracting content from PDFs and integrating it into the existing database, the entire pipeline process was completed in a total of 6.57 seconds.

This performance was evaluated against that of Unstract, which utilized a state-of-the-art GPT-3.5-based large language model (LLM) and LLM extractor, alongside a Qdrant database and an enterprise-provided LLM whisperer. As seen in Figure 14, the prompts evaluated in this study were designed to extract the minimum and maximum values of the products, identify the product with the highest purchase frequency, and ascertain the individual who purchased the largest quantity of products alongside the total expenditure incurred. For an LLM-SQL agent, these queries are adequate to efficiently traverse the entire database, regardless of the volume of data stored, and produce precise results. Nonetheless, when executed on Unstract, the majority of these prompts either exceeded the designated time threshold or yielded inaccurate outcomes due to its sole reliance on LLM not only for prompt processing but also for data analysis. Observations indicate that Unstract requires an indefinite amount of time to execute complex field-based relational operations, defined in this context as operations taking more than 100 seconds. The time ellipsoid performance plot depicting the performance of one such comparative instance, as referenced herein, is illustrated in Figure 15.

Having noticed these differences between the approach that we present and existing enterprise-ready software, we document these differences in a tabular structure as is seen in Table 1. Consequently, Unstract demonstrates limitations in delivering persuasive outcomes for multi-condition filtering. Specifically, implementing a combination of functions such as WHERE(), HAVING(), and GROUP BY()—commonly compliant with SQL—is not effectively handled. Furthermore, the accurate output of specific mathematical function combinations, including MIN(), MAX(), SUM(), AVG(), and COUNT(), poses challenges for the software,

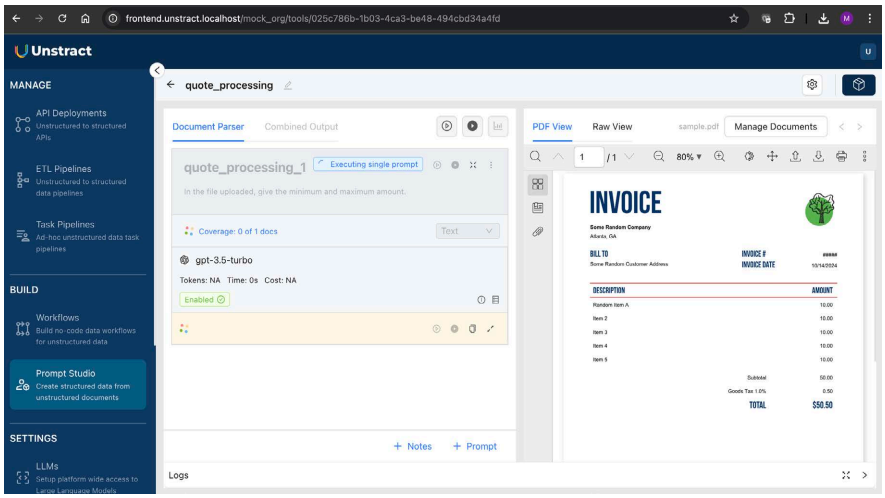

Figure 14. Unstract interface showing the LLM run for a question based on the relational data model for 1 sample document considered.

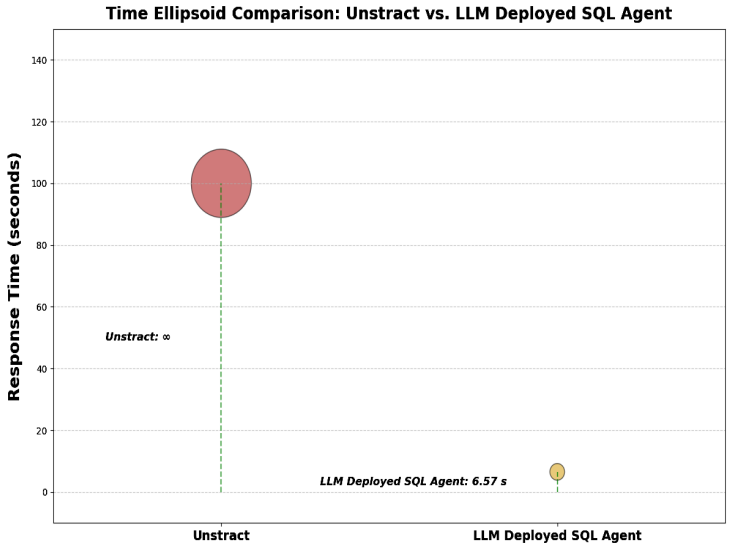

Figure 15. Time ellipsoid comparison between unstract and LLM deployed SQL agent.

Table 1. Performance comparison: unstract vs. proposed pipeline.

Metric	Proposed Pipeline	Unstract
PDF with Structured Data Extraction	<1 s	> 100 s
Query Execution	<1 s	Not Supported
Output Format	SQL	JSON
Further Operation Possible	Yes	No Field-Specific Relational Operations
Multi-Condition Filtering	Infallible	Unreliable due to permissible LLM error margin
Integration with Other Functions	Seamless due to Modular Approach	Single Purpose Operation/Lacks Flexibility
Multi-Field Data Mismatch Analysis	Efficient	Context-Trained Combined LLM Prone to Error

particularly when dealing with a single multi-paged document or multiple documents. Additionally, in scenarios where numerical and field comparisons are critical, such as in detecting inconsistencies between purchase orders and vendor quotes, Language Learning Models (LLMs) tend to be error-prone and necessitate human intervention for validation. However, an LLM-powered SQL agent is capable of supporting complex multi-field numerical comparisons, such as assessing discrepancies in price, quantity, and discount, to thoroughly analyze data mismatches.

As previously noted, the software is limited to processing PDF inputs and rendering outputs in JSON format, resulting in a significant inefficiency for subsequent operations on the database by utilizing the intermediate results. In contrast,

the proposed pipeline incorporates an SQL agent that facilitates adjustments to administrative permissions, thereby permitting modifications to the data stored within the database. More conveniently, it can be deduced from Table 2, that the level of operational flexibility that an LLM-powered SQL agent allows is not achieved with solutions that are developed leveraging the LLMs alone. Also, these solutions are always prone to certain error occurrences. Table 2 compares specific operation chains for structured data analysis which is not possible for Unstract to perform given its complete reliance on LLM alone.

Table 2. Comparison of SQL agent and unstract capabilities for complex operation chains.

Complex Operation Chain	SQL Agent Capability	Why Unstract Can't Perform
Double Join with Aggregation and Grouped Filtering	√ Executes multi-level joins with complex conditions in one query.	JSON lacks relational joins and multi-level aggregation capabilities, requiring scripting.
Nested Subquery for Conditional Summing and Ranking	√ Supports subqueries, ranking, and aggregations in a single command chain.	JSON format needs additional processing for multi-step transformations; lacks ranking capability.
Dynamic Date-Based Analysis with Conditional Joins	√ Handles time-sensitive data and complex conditionals efficiently.	JSON is flat-structured and lacks time-based processing and complex conditional joins.
Recursive CTE with Hierarchical Summing	√ Enables recursive queries for cumulative hierarchical calculations.	JSON lacks hierarchical structure, making recursion impractical.
Complex Pivot with Conditional Aggregates	√ Performs pivot operations with conditional aggregates directly.	JSON lacks pivot functionality and requires extra transformation steps for conditional aggregates.

6. Potential Under-Performance and Further Scope for Improvement

The pipeline efficiently addresses challenges associated with querying structured and unstructured data. However, certain inherent limitations could be encountered. For instance, if for an application leveraging the pipeline, the input data deviates significantly from the training distribution of the LLMs, the extracted entity or the formulated relational query may be of reduced accuracy. Such instances may arise with the inclusion of the following:

1) Input documents containing highly domain-specific language.
2) Unconventional table structures in the uploaded document.
3) Document with poor quality and ambiguous data.

Additionally, in cases of ambiguous prompts engineered for primary LLM, the accuracy of appropriate tool selection for the query may be marred.

Further scope for improvement lies in addressing these business and application-specific limitations. The techniques such as prompt engineering of primary LLM and domain-specific fine-tuning of the LLMs along with including certain fallback mechanisms such as rule-based extraction for predictable tasks, could improve the robustness of the pipeline catering to specific business applications.

7. Conclusion

We demonstrate that LLM-powered intelligent SQL, RAG, and Tavily agents markedly enhance extraction accuracy compared to existing enterprise-level solutions available in the market. With empirical evidence supporting these outcomes, this method is well-suited for broader applicability. By integrating business-specific prompt engineering of the primary LLM and accommodating flexible pipeline modifications in alignment with the application objectives, enterprises can adopt this for efficient data processing and output generation. Through minor alterations based on knowledge graphs to the control flow, this methodology facilitates the addition of another user for database interaction and can be further developed to support communication channels among users. Access to modify the database could be appropriately regulated. Furthermore, in financial contexts of structured data analysis, a hyperbolic scaling function, specifically of the hyperbolic tangent nature, could be utilized to introduce an additional field in tables, categorizing products into high, medium, and low price ranges. These represent some instance-based minor refinements, the implementation of which is contingent upon the specific application under development.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] https://planergy.com/blog/manual-procurement-process/

[2] Bahameish, B., Yaqot, M., Franzoi, R. and Menezes, B. (2022) Artificial Intelligence in Procurement: An Overview and Case Study of Qatar Foundation. *Proceedings of the International Conference on Industrial Engineering and Operations Management*, Rome, 26-28 July 2022, 722-732. https://doi.org/10.46254/eu05.20220146

[3] Yang, J., Hu, X., Xiao, G. and Shen, Y. (2024) A Survey of Knowledge Enhanced Pre-Trained Language Models. *ACM Transactions on Asian and Low-Resource Language Information Processing*. https://doi.org/10.1145/3631392

[4] Kalyanpur, A., Saravanakumar, K.K., Barres, V., McFate, C.J., Moon, L., Seifu, N., Eremeev, M., Barrera, J., Bautista-Castillo, A., Brown, E. and Ferrucci, D. (2024) Multi-Step Knowledge Retrieval and Inference over Unstructured Data. arXiv: 2406.17987.

[5] Zhou, M.Y. (2024) Improving LLM Understanding of Structured Data and Exploring Advanced Prompting Methods. Microsoft Research Blog.

[6] Biswas, A. and Talukdar, W. (2024) Robustness of Structured Data Extraction from In-Plane Rotated Documents using Multi-Modal Large Language Models (LLM). *Journal of Artificial Intelligence Research*, **4**, 176-195.

[7] Fang, X., Xu, W.J., Tan, F.A., Zhang, J.N., Hu, Z.Q., Qi, Y.J., Nickleach, S., Socolinsky, D., Sengamedu, S. and Faloutsos, C. (2024) Large Language Models (LLMs) on Tabular Data: Prediction, Generation, and Under-Standing—A Survey. https://doi.org/10.48550/arXiv.2402.17944

[8] Narayanan, P.P. and Narayana Iyer, A.P. (2024) HySem: A Context Length Optimized LLM Pipeline for Unstructured Tabular Extraction. arXiv: 2408.09434.

[9] Li, H., Gao, H., Wu, C. and Vasarhelyi, M.A. (2023) Extracting Financial Data from Unstructured Sources: Leveraging Large Language Models. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4567607

[10] Dagdelen, J., Dunn, A., Lee, S., Walker, N., Rosen, A.S., Ceder, G., et al. (2024) Structured Information Extraction from Scientific Text with Large Language Models. *Nature Communications*, **15**, Article No. 1418. https://doi.org/10.1038/s41467-024-45563-x

[11] Yang, Y., Wu, Z., Yang, Y., Lian, S., Guo, F. and Wang, Z. (2022) A Survey of Information Extraction Based on Deep Learning. *Applied Sciences*, **12**, Article 9691. https://doi.org/10.3390/app12199691

[12] Shan, Y., Lu, H. and Lou, W. (2023) A Hybrid Attention and Dilated Convolution Framework for Entity and Relation Extraction and Mining. *Scientific Reports*, **13**, Article No. 17062. https://doi.org/10.1038/s41598-023-40474-1

[13] Yang, Y., Tang, Y.X. and Tam, K.Y. (2023) InvestLM: A Large Language Model for Investment Using Financial Domain Instruction Tuning. arXiv: 2309.13064.

[14] Krugmann, J.O. and Hartmann, J. (2024) Sentiment Analysis in the Age of Generative AI. *Customer Needs and Solutions*, **11**, Article No. 3. https://doi.org/10.1007/s40547-024-00143-4

[15] Parthasarathy, V.B., Zafar, A., Khan, A. and Shahid, A. (2024) The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities. arXiv: 2408.13296.

[16] Trad, F. and Chehab, A. (2024) Prompt Engineering or Fine-Tuning? A Case Study on Phishing Detection with Large Language Models. *Machine Learning and Knowledge Extraction*, **6**, 367-384. https://doi.org/10.3390/make6010018

[17] Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J. and Wu, X. (2024) Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering*, **36**, 3580-3599. https://doi.org/10.1109/tkde.2024.3352100

[18] Hello, N., Di Lorenzo, P. and Strinati, E.C. (2024) Semantic Communication Enhanced by Knowledge Graph Representation Learning. 2024 *IEEE 25th International Workshop on Signal Processing Advances in Wireless Communications* (*SPAWC*), Lucca, 10-13 September 2024, 876-880. https://doi.org/10.1109/spawc60668.2024.10694291

[19] Zhao, H., Jiang, W., Deng, J., Ren, Q. and Zhang, L. (2023) Constructing Knowledge Graph for Electricity Keywords Based on Large Language Model. 2023 *IEEE 7th Conference on Energy Internet and Energy System Integration* (*EI2*), Hangzhou, 15-18 December 2023, 4844-4849. https://doi.org/10.1109/ei259745.2023.10512525