Research Project Proposal

AdvanHR

Enhancing HR System with Advanced Conversational and Generative AI

Department of Computer Science

University of Engineering and Technology, Taxila

Introduction

In today's fast-paced business environment, efficient human resource management is crucial for organizational success. ***"AdvanHR,"*** **a Microsoft-registered startup,** is an innovative digital solution designed to enhance HR operations through automation, advanced analytics, and intelligent processing. This project integrates several key functionalities that empower HR professionals and improve employee engagement.

The **Virtual Assistant** serves as a central tool for automating routine HR tasks by converting natural language inputs into database queries, retrieving relevant information, and presenting it in an easily understandable format. Our **Analytics** module offers both user-driven and AI-driven insights, enabling HR personnel to perform customized data analysis. In the user-driven component, HR professionals can articulate specific analytical needs in natural language, which in turn provides the analysis required by the HR. In contrast, the AI-driven component automatically detects important trends and meaningful analyses within the database, presenting these insights proactively to support data-informed decision-making.

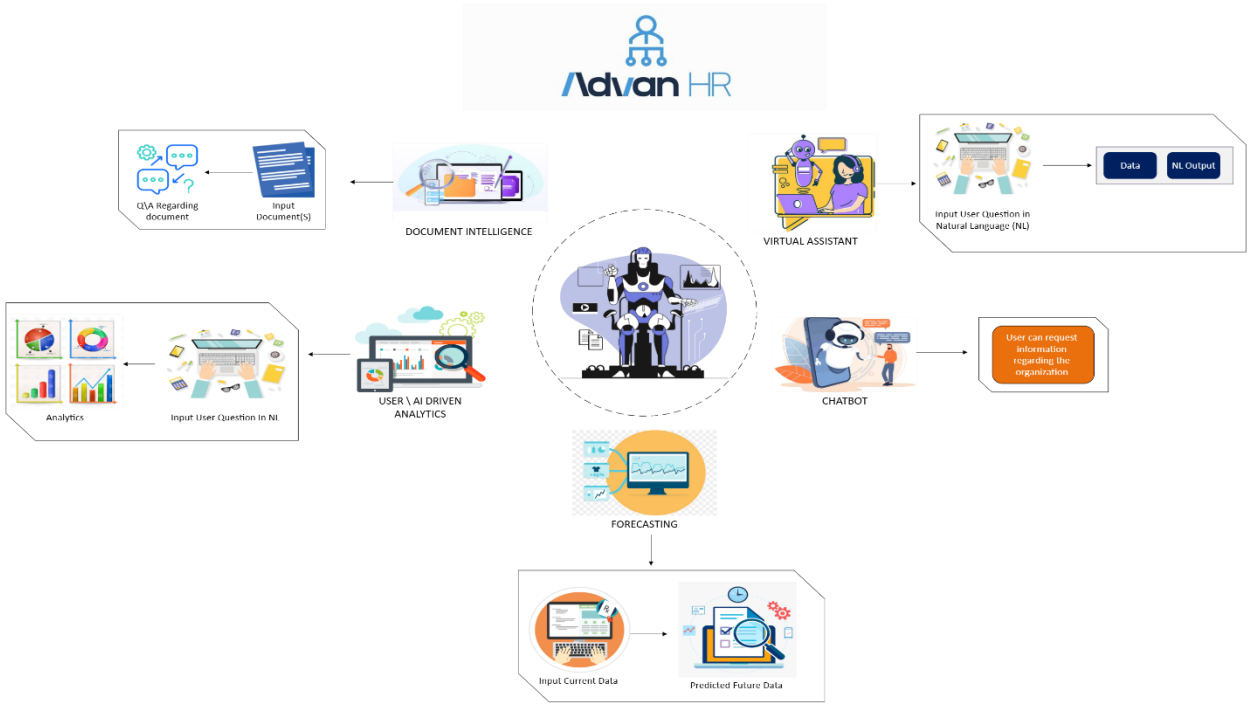
With a focus on predictive capabilities, the **Forecasting** feature helps organizations anticipate employee turnover and sales trends, facilitating proactive decision-making. The **Intelligent Document Processing** function analyzes and interprets documents, allowing users to extract specific information or receive summaries based on their queries.

Additionally, the **Chatbot** component acts as an employee self-service tool, providing instant support and information to staff members, thereby enhancing overall workplace efficiency.

By combining these functionalities, AdvanHR aims to streamline HR processes, foster data-driven decisions, and elevate employee experiences within organizations.

AdvanHR

AdvanHR is a user-friendly web application that offers a wide range of functionalities, as outlined above. Presented below is a visual representation of AdvanHR's features to provide a clearer understanding of its capabilities.



The sections below delve into each core module of AdvanHR, highlighting their individual roles, underlying technologies, and the impact they bring to modern HR management.

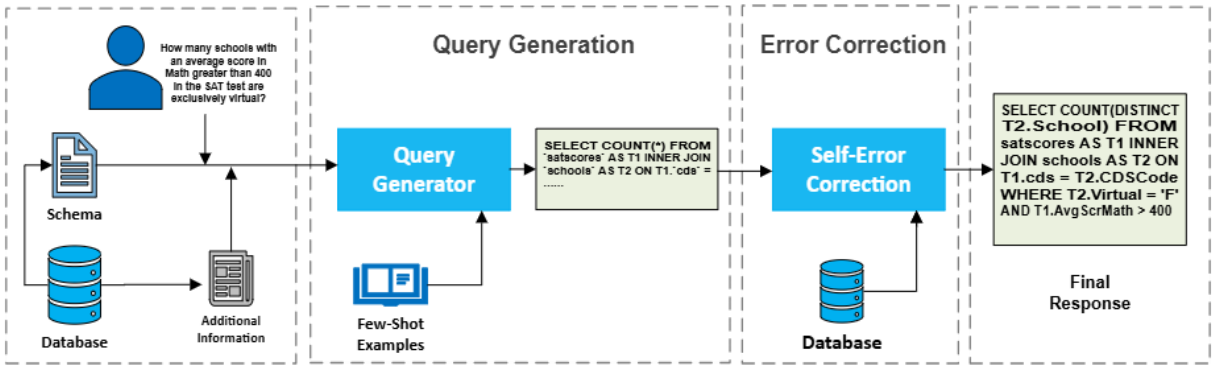
Virtual Assistant

In the Virtual Assistant module, the user inputs a question in natural language and receives a response comprising both the required data and a natural language explanation. Below is a technical overview of how this functionality is implemented.

For query generation from natural language, which is the most crucial component of this module, we have proposed two approaches: LangSQL-mini for quick query generation and execution with low resource consumption, and LangSQL with additional internal processes, offering improved accuracy at the cost of higher resource usage and comparatively longer processing time than the LangSQL-mini. We have leveraged Microsoft Azure's cloud platform for seamless model access, ensuring reliability, and secure integration of AI services.

A. LangSQL-min

LangSQL-mini unlike the LangSQL, does not involve many internal steps. The methodology of LangSQL-mini is shown in figure below and explained as follows:



The process begins with retrieving the schema of the connected database. This schema is essential as it serves as a foundational part of the prompt template, which guides the query generation. Alongside the schema, if the connected database contains any additional information such as evidence or other informative files, this data is included in the prompt template to provide context. Furthermore, a set of few-shot examples is added to the template. These examples, which cover three different levels of difficulty (simple, moderate, and complex), act as references to demonstrate how queries should be structured for various question types.

Once the prompt template is fully prepared, it is passed to a LLM by invoking a chain. The LLM analyzes the template, including the user question, schema, additional information, and examples, and generates an SQL query tailored to the given input. This generated query is then executed on the database to retrieve the required results.

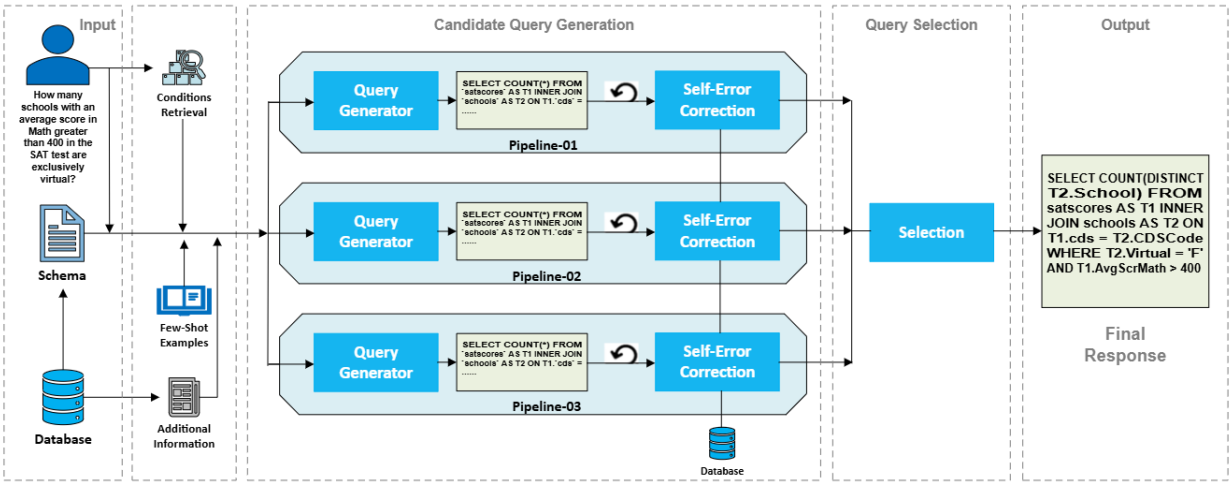
After execution, the results are reviewed by an error correction agent. This agent checks whether the execution encountered any errors or returned null value. If no errors occur and the query successfully retrieves the desired results, the query generated in the first attempt is finalized as the correct query. However, if any issues are detected then the error message and the previously generated query are sent back to the LLM for reprocessing. The LLM then generates a revised query to address the identified problems, and this new query is selected as the final query.

The steps above are implemented through a series of pipelines for seamless execution.

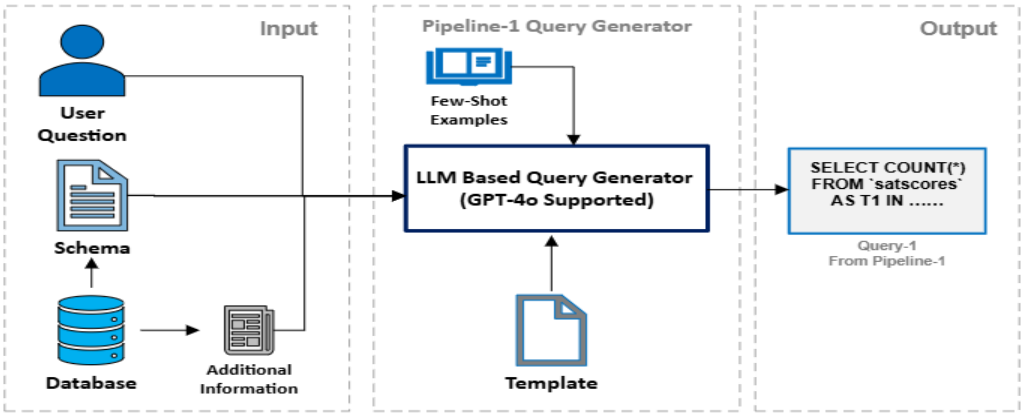
In this way, LangSQL-mini is implemented with very simple architecture, low resource consumption, and fast processing.

B. LangSQL

LangSQL consists of several internal steps, which are illustrated in figure below.
This architecture consists of three pipelines: Pipeline-1, Pipeline-2, and Pipeline-3, and a candidate selection agent. The first three pipelines are used to generate candidate SQL queries by using different internal procedures.



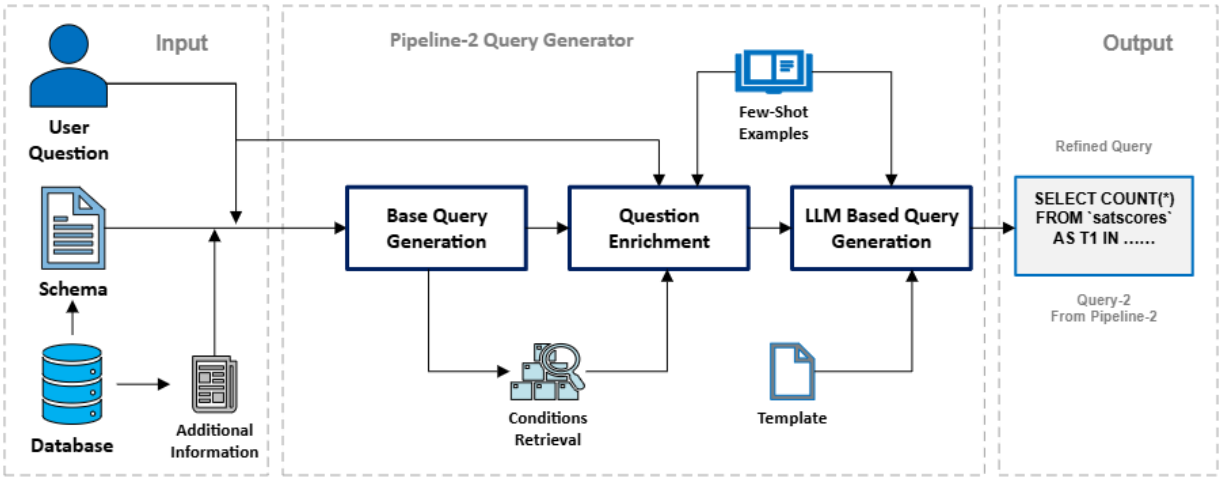
Pipeline-1: Pipeline-1 is depicted in Figure below. This pipeline generates a query using Lang-SQL-mini. However, unlike the LangSQL-mini where the query undergoes the error correction agent only once, in this approach, the query can undergo the error correction agent a maximum of two times, depending on the need. If, after the first iteration, the error persists or a null value is returned, the query is reprocessed through the agent a second time. The error correction agent itself functions similar to the one used in the LangSQL-mini.



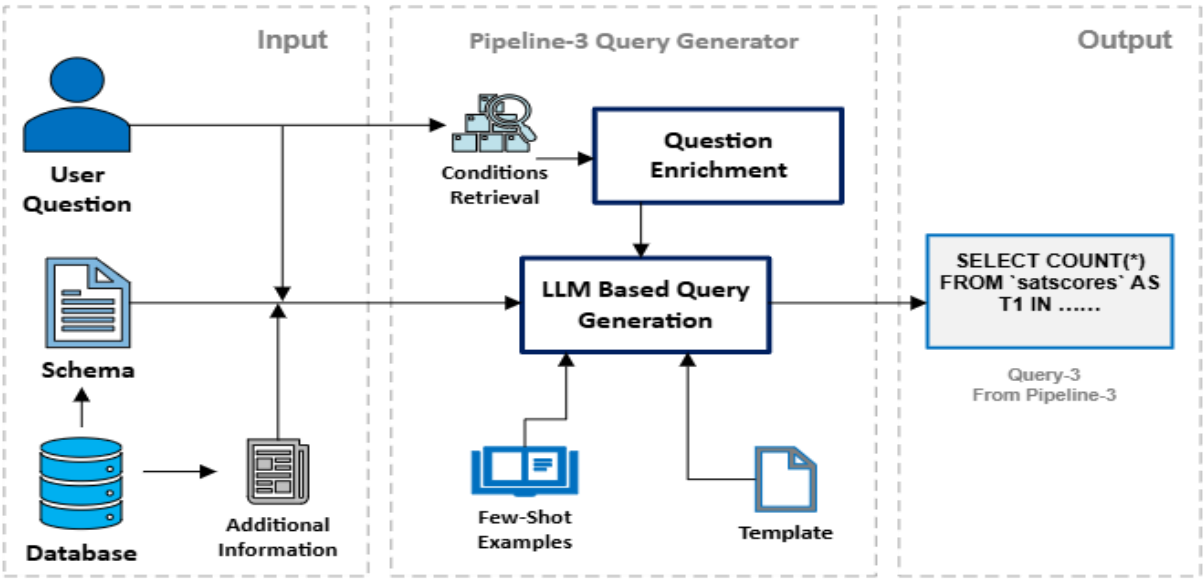
Pipeline-2: Pipeline-2 is illustrated in Figure 4. In this pipeline, the SQL query is initially generated using a simple candidate SQL query generation template. After the query is generated, conditions such as WHERE, JOIN, ORDER BY, etc., are extracted from the generated query. These conditions are then used by the question enrichment template, along with the database

schema, few-shot examples, or any other additional information provided for the target database, to enrich the user question. The SQL query is then generated from this enriched question.

After this, the SQL query may undergo the error correction agent a maximum of two times if needed. In this pipeline as well the error correction agent functions similar to the one used in the LangSQL-mini.



Pipeline-3: An illustration of pipeline-3 is provided in figure below. In this pipeline, conditions from the user's question are first retrieved (e.g., if the question contains "WHERE salary=3000"). These conditions are then used to enrich the question using a question enrichment template, which incorporates the database schema, few-shot examples, and any additional information provided for the target database. The SQL query is then generated from the enriched question using the candidate SQL query generation template. Subsequently, the query may undergo the error correction agent, following the same process as implemented in Pipeline-2.



In this way, candidate queries are generated by the three pipelines. These queries are then passed to the Candidate Selection agent.

Candidate Selection agent: In this step, the embeddings for the question, database schema, and additional information are calculated and then multiplied by their respective weights: 40% for the question, 30% for the schema, and 30% for the additional information. The results of multiplication are then added together to form a cumulative embedding.

In parallel the embeddigs for three candidate SQL queries is calculated separately.

Subsequently, the cosine similarity between the cumulative embedding and the embeddings of candidate SQL queries is calculated separately. The query with the highest cosine similarity score is then selected as the final query.

Given two vectors A and B, the cosine similarity between them is calculated as

Cosine Similarity *= (A.B) / (||A|| * ||B||)*

Where:

- **A. B** is the dot product of vectors **A** and **B**.
- **||A||** and **||B||** are the magnitudes(lengths) of vectors **A** and **B** respectively.

Once the query is finalized using one of the two models, the resulting data is passed to the LLM for the final natural language response generation. A threshold has been set such that if the output generated by the LLM exceeds 50 tokens, the system summarizes the response. This summary is based on both the data retrieved by the final query and the user's original question. The threshold is configurable and can be adjusted or removed entirely, depending on the preferences of the system user. This mechanism was implemented to enhance resource efficiency.

Furthermore, both of our architectures, **LangSQL** and **LangSQL-mini**, were evaluated on the BIRD benchmark as well as the Spider dataset. On the BIRD benchmark, LangSQL and LangSQL-mini achieved accuracies of **76.72%** and **72.94%**, respectively. Notably, LangSQL outperformed existing systems on the BIRD benchmark. On the Spider dataset, LangSQL achieved an impressive accuracy of **92.27%**.

Document Intelligence

In Document Intelligence module the user can upload document in multiple formats, such as PDF, DOCX, and TXT. Upon upload, the system utilizes specialized Python libraries and functions to extract content from it. When the content is extracted, it is then divided into chunks, each with a size of 500. Let C represent the extracted content from the document.

The chucking process will be formulated as:

$$C=\{c1,c2, \dots ,cn\} \text{ where } |ci| \leq 500$$

Where **ci** represents an individual chunk and **| ci |** denotes its word count.

After chunking the content ci, embeddings E(ci) are generated for each chunk using the Embedding-3 Large model.

$$\mathbf{E(ci) = f(ci),\ E(ci) \in R^d}$$

Here f is the embedding function, and d is the embedding dimension.

This model understands context better than previous versions and is highly effective with minimal labeled data, making it ideal for zero-shot or few-shot learning.

Moving forward, both the content and embeddings are indexed in the Azure AI Search Service. In Azure AI Search, an index is created with three fields:

1. **id** (of type emd.string)
2. **content** (of type emd.string)
3. **embeddings** (of type emd. collection(single))

The extracted content and corresponding embeddings are then stored in these fields. Once the data is indexed in the Azure AI Search Service, the user can ask any question(s) they wish.

Once the user has entered their question(s), Azure Search Service retrieves the relevant chunks.

The service first generates embeddings (Embeddings are numerical vector representations of data such as words, sentences, images, or documents in a continuous vector space) for the user question(s) using the same model that was used for creating embeddings of the chunks. Then, **semantic search** is performed by **Azure AI Search Service** to retrieve the most relevant chunks. If multiple questions are provided, the service retrieves relevant chunks separately for each one. In total, the top 10 chunks are retrieved for every question depending upon the similarity between the embedding of the user question and the embedding of the chunk.

Once the chunks are retrieved by Azure Search Service for the user's question(s), any duplicate chunks are removed, which may occur in the case of multiple questions. After removing duplicate chunks, external reranking is performed to ensure a more precise and accurate response generation.

In external reranking, cosine similarity is used to identify the chunks that are most relevant to the user's query. The similarity score is calculated between the user question(s) and the retrieved chunks. In the case of multiple questions, the similarity score is determined for each question against every retrieved chunk. Each chunk is then assigned the highest similarity score it receives from different sub-questions. After assigning scores to each chunk, the top k chunks with the highest scores are selected as the final chunks. In our system, the value of k is set to 7.

Cosine Similarity measures the similarity between two vectors in multi-dimensional space by calculating the cosine of the angle between them. The closer the cosine similarity is to **1**, the more similar the vectors are.

Given a user query Q, its embedding is computed as:

$$\mathbf{E(ci) = f(ci),\ E(ci) \in R^d}$$

The relevance of each chunk is computed using **cosine similarity:**

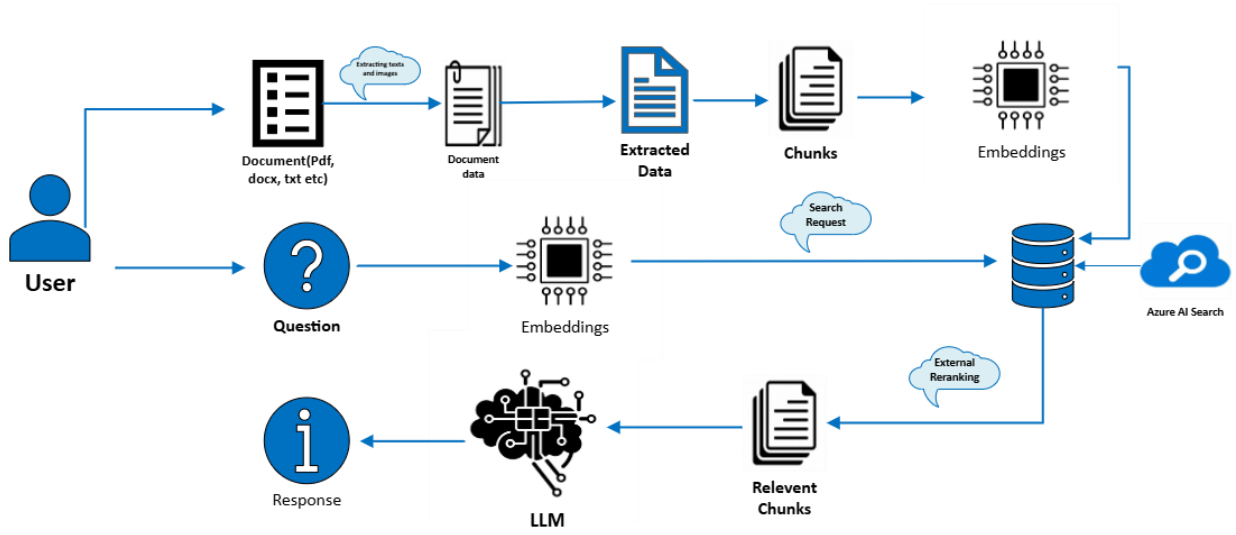
$$\textbf{\textit{Cosine Similarity}} = E(Q).E(ci) / \ |E(Q)| \times |E(ci)|$$

Where:

- $E(Q).E(ci)$ Is the dot product.
- |E(Q)| and |E(ci)| are the magnitudes of the vectors.
- The top 7 chunks with the highest similarity scores are retrieved.

Moving forward, the final chunks along with the user question(s) are passed to the GPT-4 omni to generate the response.

External reranking is effective in terms of resource utilization and accuracy, because the shorter the prompt will be to the LLM the smaller the number of tokens is given as input, and the LLM will be able to understand the context better because there will be no irrelevant information concerning the user question(s).



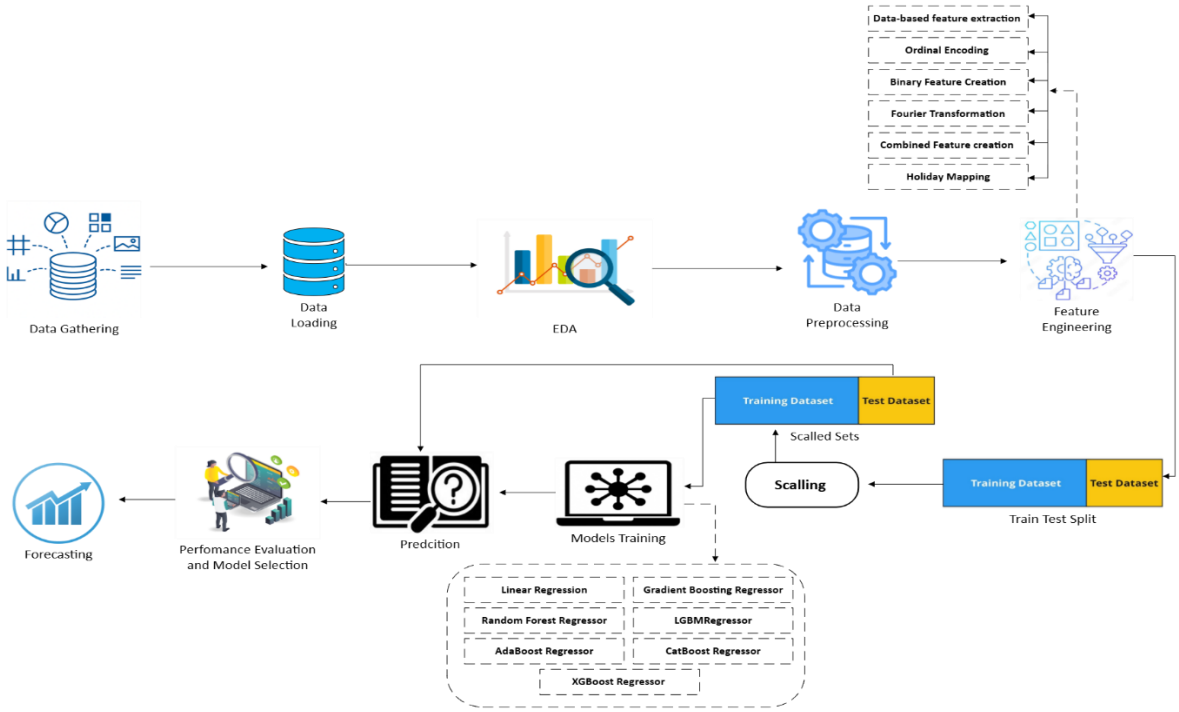
A research paper based on this module is also nearing publication, as it is currently at the video-ready stage. The accuracies achieved by this module using different models, as detailed in our research paper, demonstrate its effectiveness and robustness.

Forecasting

For our forecasting module, the process begins with data collection. Approximately 600,000 records are gathered to train the models used in this module. Once the data is collected, it is loaded into the system and subjected to Exploratory Data Analysis (EDA) to examine various features, assess their relevance, and determine which features should be retained, removed, or combined to generate new, more informative features.

Following this, data preprocessing is carried out, which involves handling missing values, ensuring data consistency, and performing several other cleaning steps. After preprocessing, a series of feature engineering techniques are applied, including:

- **Data-based feature extraction** – Deriving meaningful attributes from raw data, such as extracting the day of the week or month from a date field.
- **Ordinal encoding** – Converting categorical variables with a natural order into numerical values.
- **Binary feature creation** – Creating features with binary (0 or 1) values to indicate the presence or absence of a certain condition.
- **Fourier transformation** – Applying Fourier analysis to capture seasonality and cyclic patterns in time-series data.
- **Combined feature creation** – Generating new features by combining two or more existing ones to capture complex relationships.
- **Holiday mapping** – Incorporating national or regional holidays into the dataset to model their effect on sales or demand.



Once feature engineering is completed, we split the data into training and testing sets. As previously mentioned, the training data set consists of 600,000 records. Both the training and testing sets are then scaled appropriately, and the training data is used to train the models.

A range of models are explored for forecasting, including:

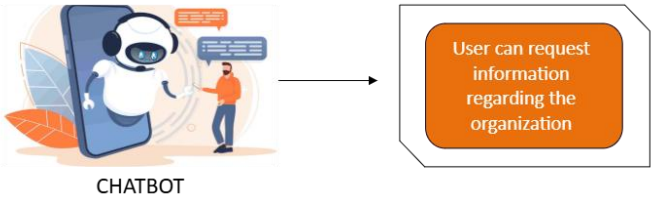
- **Linear Regression**

- **Gradient Boost Regressor**
- **Random Forest Regressor**
- **CatBoost Regressor**
- **AdaBoost Regressor**
- **XGBoost Regressor**

Once the training is complete, the models are tested, and the one that outperforms the others is selected as the final model, which is then used for forecasting

Chatbot

In our chatbot module developed within Azure AI Studio, we created an assistant designed to provide support related to the organization's policies. We prepared and uploaded documents covering various types of company policies, enabling the assistant to assist users by offering relevant and accurate information based on the documentation provided.

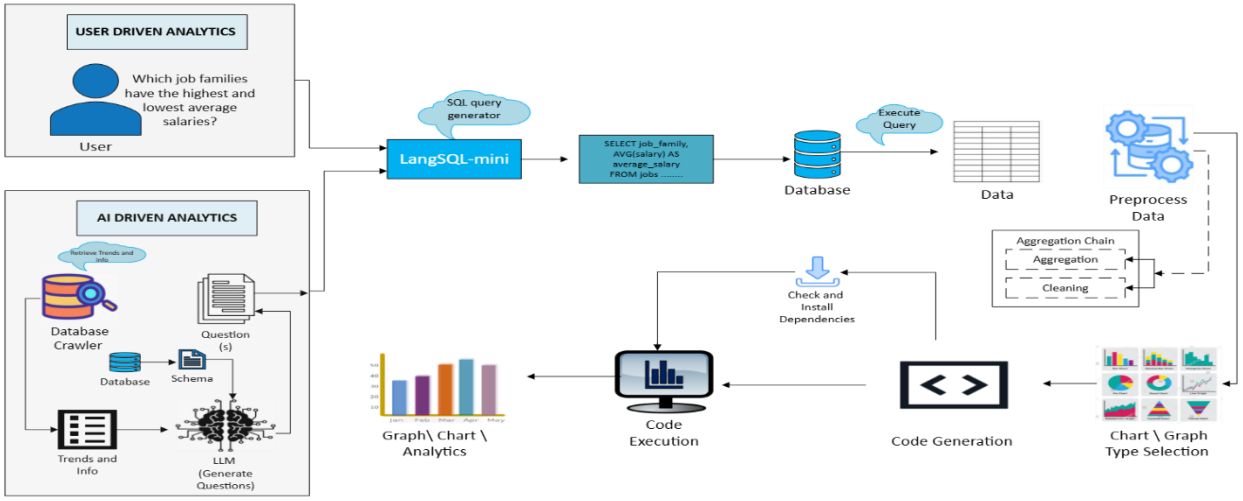


User\ AI Driven Analytics

In this module, users are provided with two options: **User-Driven Analytics** and **AI-Driven Analytics**.

- **User-Driven Analytics**: In this mode, the user manually inputs a query in natural language. This query is then processed by the system to retrieve the desired insights.
- **AI-Driven Analytics**: In this mode, the system autonomously analyzes the database to identify significant trends and relevant information. These insights are then passed to a language model, which generates a set of meaningful questions for the user. The user selects one of these questions to proceed.

Once the input question (from either mode) is finalized, it is passed to **LangSQL-mini**—a lightweight version of LangSQL used to optimize resource consumption. LangSQL-mini converts the natural language query into an executable SQL query, which is then used to retrieve data from the database.



The retrieved data is passed to a **preprocessing stage**, where it undergoes aggregation and cleaning to ensure quality and relevance.

Following preprocessing, the system determines the most appropriate **chart type** based on both the user's question and the nature of the data. After selecting the chart type, **GPT-4o** is used to generate the corresponding chart code.

If any required libraries or dependencies are not already installed on the system, they are automatically installed at this stage. Once the code is ready, it is forwarded to the **execution module**.

The execution module runs the generated code and produces the final **visual chart** as output for the user.

Job Interview and Resume Screening

- Filters and shortlists resume based on predefined job requirements, ensuring that only qualified candidates proceed to the next stage of the recruitment process
- Performs initial technical assessments by calling the candidates.