Project report on

AI Financial Advice Chatbot - The FinSavvys Team

Group 9: Sasi Vasa - 801223654, Sudeepta Bal - 801455628, Mitali Yadav - 801453849, Rucha Tatawar - 801420899 and Sahit Ceeka - 801424751

API Dependencies involved in the Project

Our project combines the Groq API for LLM-based transaction classification and financial information production. However, the team's present accessibility is limited to a free-tier Groq account, which restricts the number of allowable API queries per day. Because these credentials are tied to an individual teammate's account and cannot be publicly shared, the system cannot be fully deployed or hosted publicly at this time. This API dependency impacts our ability to execute the model continuously, making large-scale testing and public hosting difficult under current access constraints.

Introduction, Problem Statement, Objectives

The Personal Finance Chatbot **helps many users understand their spending without manually going through long bank statements. Many people struggle with organizing expenses, tracking habits and building budgets. When users don't have a clear understanding of their actual spending, budget planning becomes difficult and results in unattainable or inefficient budgetary goals. Users find it challenging to ask precise concerns about their finances because most financial apps just offer static displays without interactive explanations.** The system **solves this by letting users upload their financial files, automatically extracting transactions, categorizing them and giving clear summaries through a chat interface.**

Objectives:

- Extract transaction data from PDF, CSV or Excel files.
- Clean and standardize financial data.
- Categorize expenses automatically.
- Analyze spending patterns and highlight trends.
- Generate realistic monthly budgets.

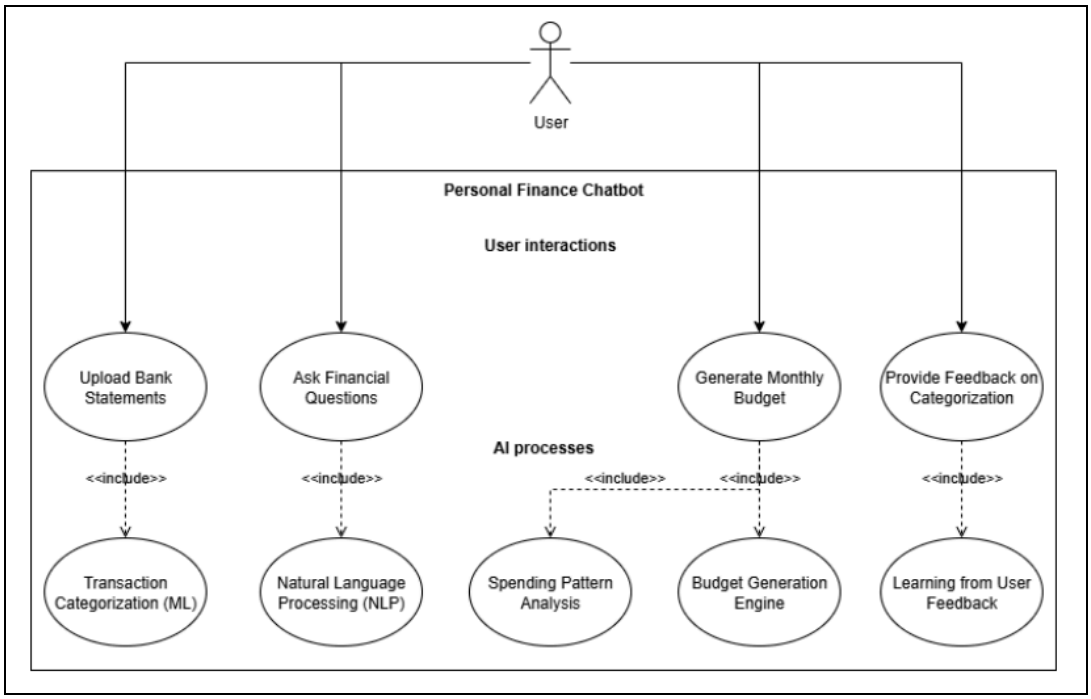
- Answer financial questions in natural language.
- Update calculations in real time when transactions change.

AI Methodology & Algorithms Used

The system uses a simple AI pipeline where uploaded financial files are first cleaned and standardized using Python routines, fixing headers, dates and missing values. **Once the data is prepared, a large language model (accessed through the Groq API)** is used to categorize transactions because it handles vague or inconsistent descriptions better than earlier attempts with Random Forest or XGBoost. **The same LLM interprets user questions, generates summaries, compares spending with the user's budget and provides recommendations through an agent-style reasoning flow.** Each time t**he user adds new transactions** or updates information, **the model re-evaluates the data and refreshes the analysis in real time**.

System Design (UML, Architecture, Data Flow and Project Requirements)

UML



System Architecture

Frontend:

- React interface showing uploads, chat messages and summaries.
- Handles user input, file upload and displays AI responses.

Backend:

- Python environment that extracts data, cleans it, categorizes transactions, analyzes spending and computes budgets.
- Communicates with an LLM for classification and reasoning.
- Updates results whenever new data is added.

Data Flow

1. The user uploads a PDF/CSV/Excel file.
2. Backend extracts dates, descriptions and amounts.

3. Cleaning corrects headers, formats and missing values.
4. AI classifies each transaction into categories.
5. The system analyzes totals, trends and recurring expenses.
6. Budgets are generated based on income and past spending.
7. The user asks questions; chatbot responds using processed data.
8. New transactions or feedback trigger immediate recalculation.

Project Requirements

A. Functional Requirements

- Automatic extraction of transaction data.
- Transaction categorization into groups like food, rent, utilities, entertainment.
- Spending pattern analysis and detection of habits or overspending.
- Ability to answer questions (totals, comparisons, category insights).
- Budget creation based on income and past expenses.
- Alerts when spending exceeds budget.
- Support for adding new transactions through chat.
- Ability to produce summaries and breakdowns on demand.
- Continuous improvement using user feedback.
- Secure handling of financial files.

B. Non-Functional Requirements

- **Performance:** There is a provision for fast responses with no significant delay
- **Scalability:** Must support multiple users and large datasets.

- **Usability:** Simple interface usable by all technical levels.
- **Reliability:** Accurate financial outputs with minimal downtime.
- **Maintainability:** Modular structure allowing easy updates.
- **Integration:** Ability to integrate with external tools (e.g., charts).
- **Data Consistency:** Synchronized, up-to-date financial information.
- Security & Privacy: Encrypted handling of financial data; no sharing without consent.

C. System Requirements

System Capabilities:

- Convert uploaded files into structured transaction tables.
- Normalize headers, formats and missing entries.
- Perform agent-style reasoning for summaries and recommendations.
- Recalculate results dynamically when new data appears.
- Store and use feedback to improve classifications.

System Constraints:

- Requires consistent formatting after cleaning to avoid model errors.
- Requires stable backend–frontend communication.
- Requires secure storage of API keys for AI functionality.

D. Hardware Requirements

- Standard computer with a CPU (no GPU needed).
- At least **8 GB RAM** to support data processing.
- Modern operating system (Windows/macOS/Linux).

E. Software Requirements

Backend Software:

- Python (suitable version for data processing).
- Libraries for extraction & processing (pandas, numpy, pdf-processing tools).
- Environment variable support for secure API key storage.
- Ability to run the chatbot's reasoning and cleaning logic.

Frontend Software:

- Node.js for running the React interface.
- Required JavaScript dependencies and styling tools (e.g., Tailwind).
- Browser-based rendering (Chrome/Edge recommended).

General Software:

- Modern browser with JavaScript enabled.
- Local or hosted environment capable of running both backend and frontend services.

F. Security Requirements

- Sensitive information must be sanitized during extraction.
- API keys stored securely in environment variables.
- No data shared externally without user consent.
- Encrypted handling of uploaded statements.

G. Data Requirements

- Must support PDF, CSV and Excel financial statements.
- Must extract date, description and amount fields.

- Must maintain clean and consistent tables for analysis.
- Must store user feedback for future improvements.

Implementation Details

The whole project is executed with the help of three combined components. Together, they help in enabling a complete end-to-end workflow:
- interactions between users and uploading file segments,
- backend financial calculations, and
- showcasing and verification.

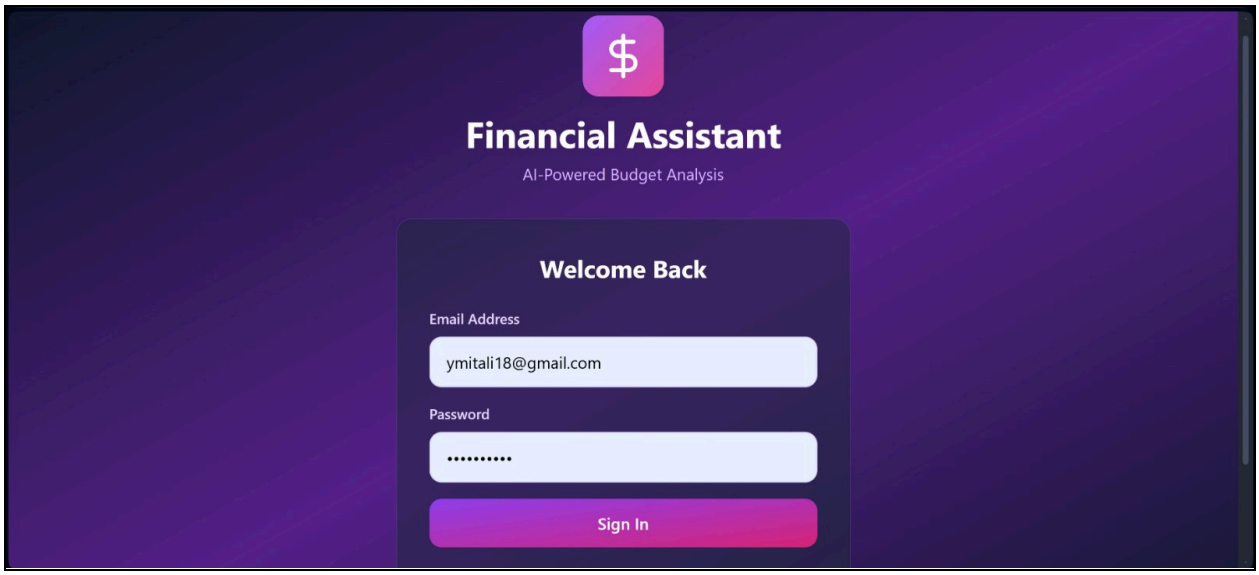
Below given is an elaborated explanation of each of the functionalities:
1. Front-End Chat Interface

The code for the front-end interface outlines how exactly the chatbot interaction with the user happens, how the chatbot handles the file uploads and displays the appropriate output/messages in a conversational format.

- Interaction between Users and the Chat flow

The interface is developed to simulate a conversation that looks more natural with a financial assistant. It
- showcases user messages and the responses of the assistant with appropriate layout.
- scrolling automatically to display the most recent reply.
- displays loading animations like the typing indicators to simulate the delay caused in AI replies.
- Avoids the submission of empty messages.

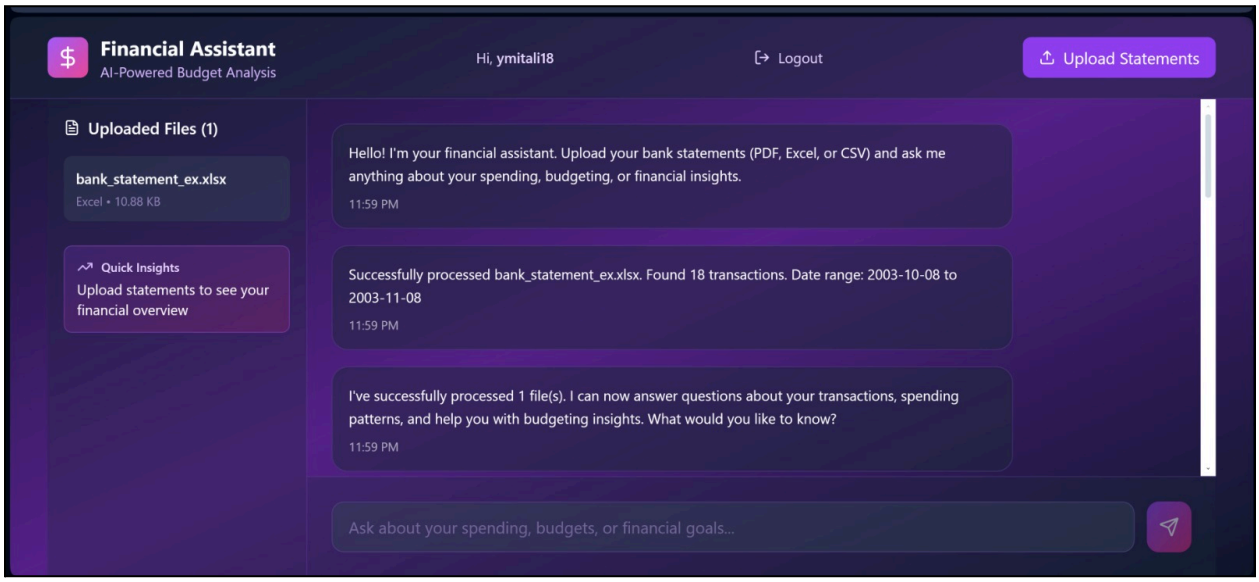
This helps in creating a smooth and interactive experience with respect to chat.



- Uploading files and Validation workflow

The UI has the feature for supporting the upload functionality of the documents with types .pdf, .csv, .xls, and .xlsx.

For each of the file uploaded, the interface showcases:
File Name, File size, File type, and the inclusion of status indicators like “processing” or “ready”
The PDF gives us the clear idea that the front-end makes sure that only the legit formats are passed on to the next step. The incorrect uploads are stopped before it reaches the backend.



Response Simulation

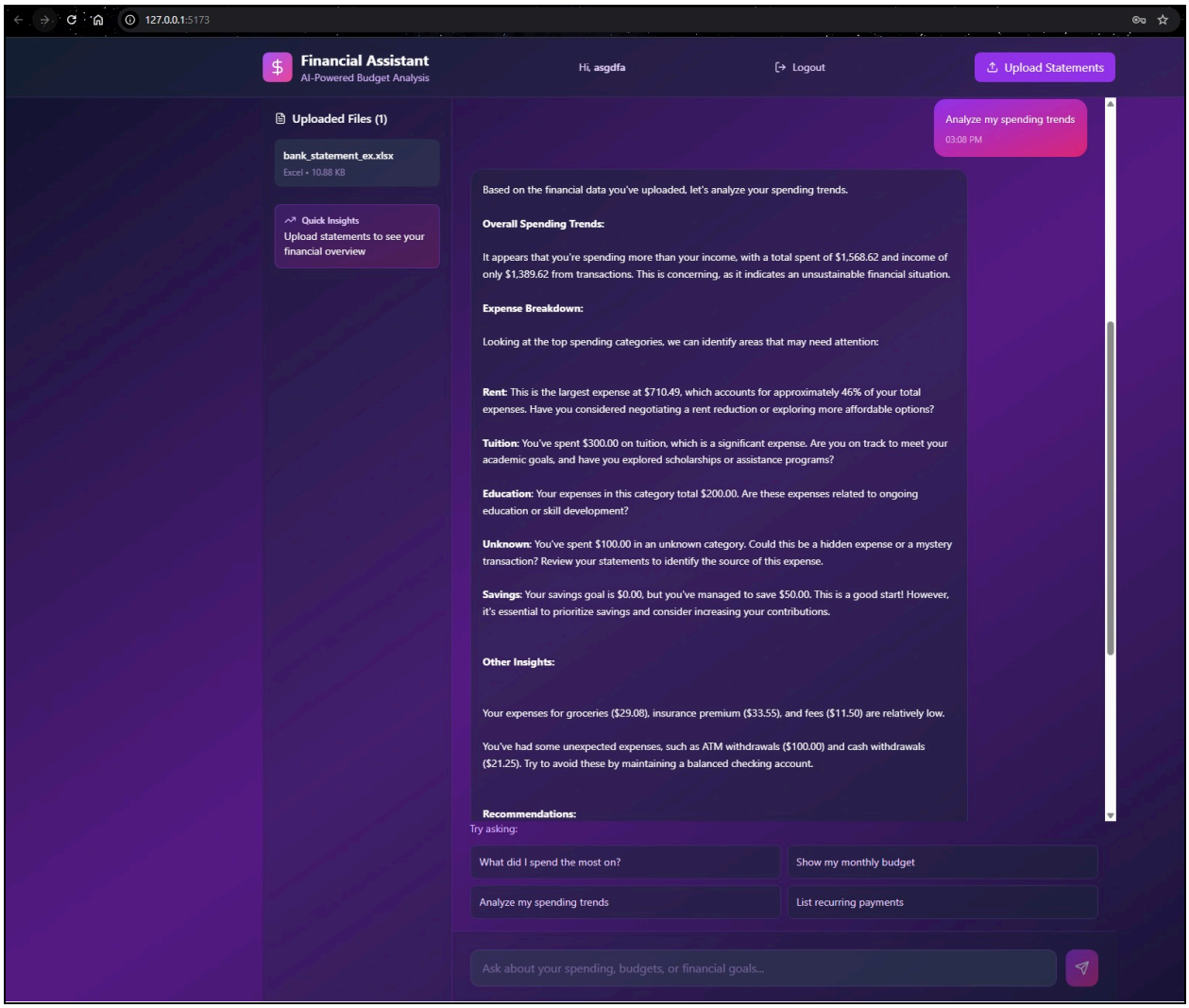
While the backend code has been developed separately, the PDF showcases that the front-end leverages predefined or simulated responses to replicate
- Analysis on Spendings
- Budget related information
- Summaries on Transactions
- Comments on Monthly Trend

This helps the UI to demonstrate a real-time experience even if the backend is not live-connected.

Visual feedback

The interface gives us-
- Color-coded bubbles with messages in it
- Loading bars
- Rendering of the Dynamic text
- Conversational history within a clean layout

This makes sure to enhance the user interactions and makes the financial advice more easier to understand.



2. Backend Financial Processing Engine

The Python code executes all the logic from the finance side, such as the budgeting, analysis done on spendings, categorization, and finally generating summaries. It does not contain any code for the front-end interface but its main purpose is entirely analytical.

Data Handling and Storage

Two pandas DataFrames are actually taking care of all transactional data
- self.transactions_df - historical expenditure loaded during initialization
- self.monthly_transactions - new transactions that can be added during user engagement.

These both are actually joined and considered together during the analysis phase to ensure no data is missed.

Income and Handling Budget

Implemented functionalities such as:
- set_income(),
- set_budget_goals(), and
- calculate_budget_allocations() helps the user to configure the income ordered month_wise and the standard budget ratios (like the 70% expenses, 20% wants and 10% savings)

The backend is responsible to normalize the budget percentages in an automated way, if they do not sum up to 100 by default, avoiding the calculation mishaps.

Transaction Classification based on LLM

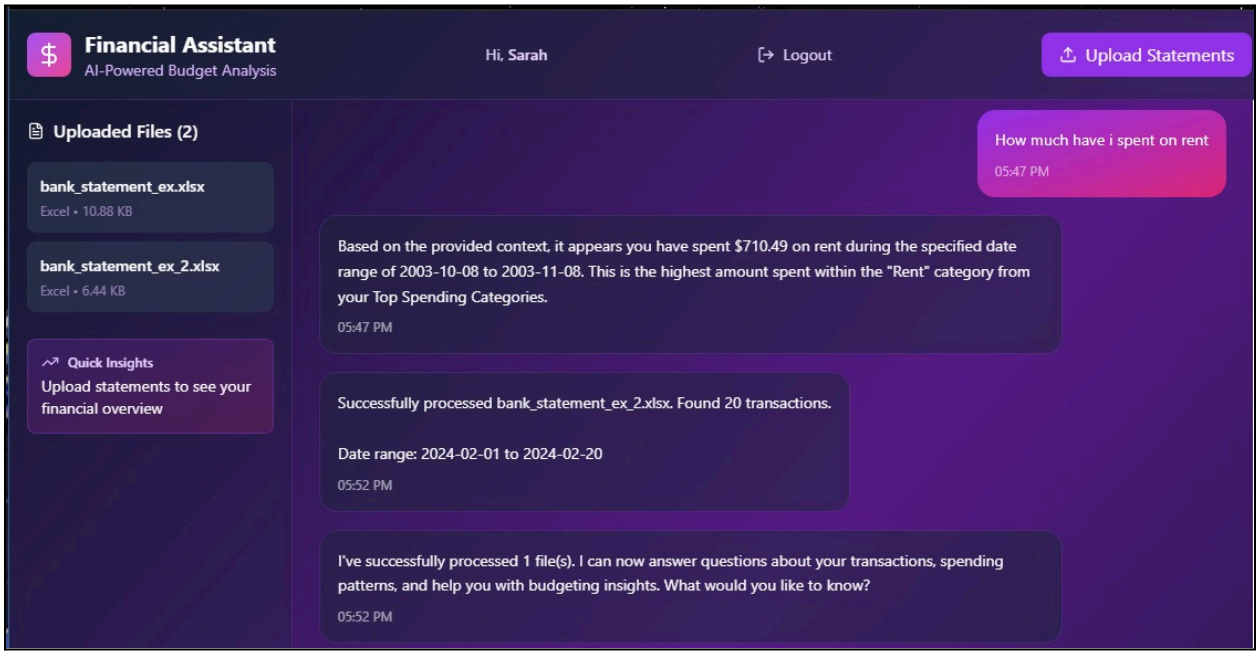
The backend logic leverages a Groq LLM API to categorize the transactions into:
- a classification (e.g., Groceries, Rent) and
- a classification type (e.g., expenses, wants and savings)
This actually takes place with the help of functions such as
- _classify_transaction()
- categorize_transactions()
The execution consists of:
- A mandatory requirement of JSON-only prompt to regulate the output
- Parsing using the try/except block to avoid the failure
- Fallback classifications ("Uncategorized or default "expenses) when the model output is incorrectly formed
This makes sure that all the transactions can still be considered even when the LLM replies vary.

Financial Explanations generated by LLM

Apart from the categorization, the backend logic also leverages the LLM to generate
- Customized financial advice
- Spending information
- Tips on improving budgets
This actually happens via _generate_response(), where the backend considers together:
- computed spendings
- Leveraged budget
- Income as per user definition
- Classification breakdowns
in the form of a prompt that is sent to the LLM. The model then returns a conversational elaboration that the UI can in turn send back to the user.

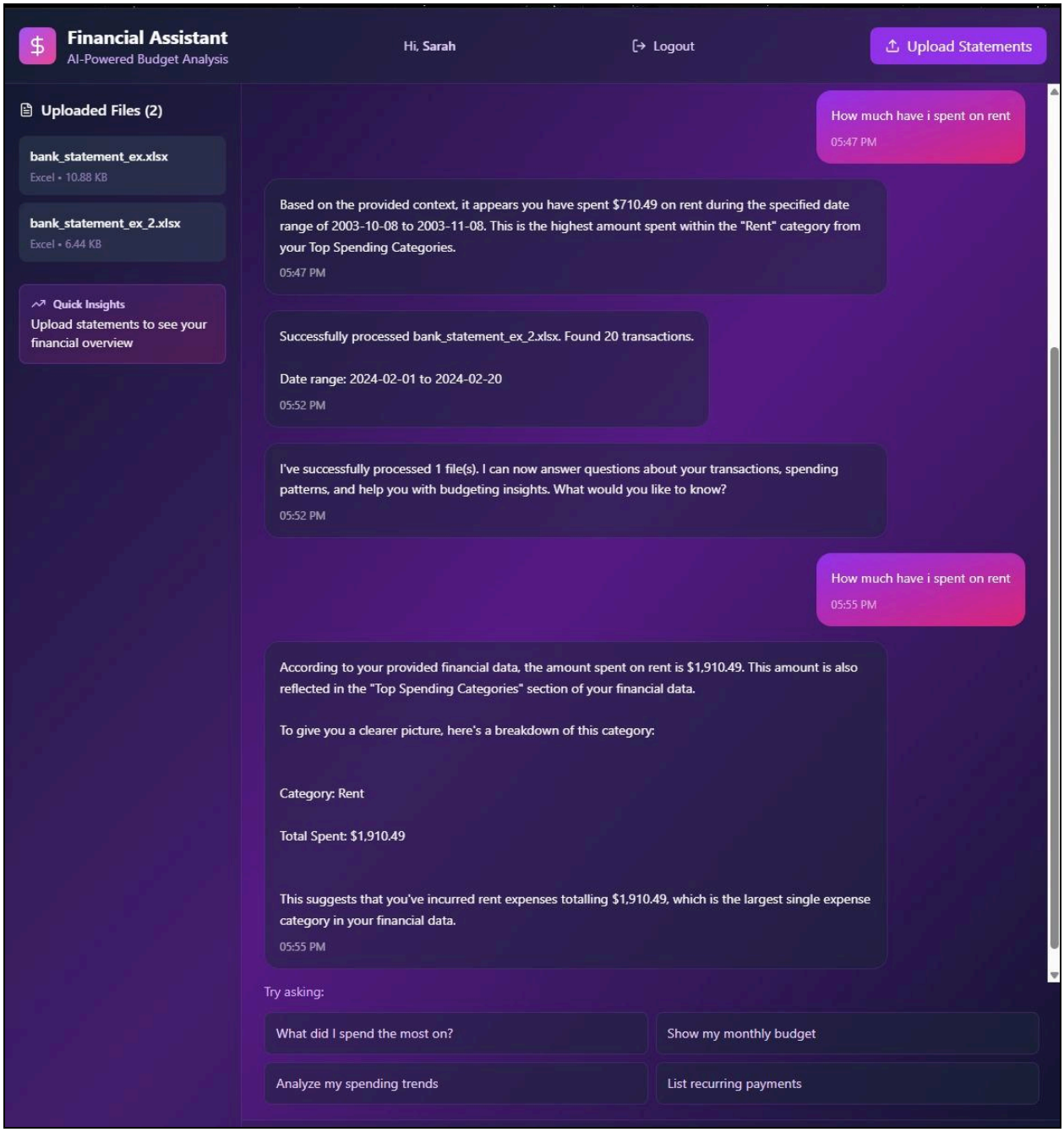
Logic implemented for Spending Analysis

The core logic is implemented with the help of the function analyze_current_spending():
- Loops through all the transactions
- Sums up the amount by category_type (expenses or wants or savings)
- Develops a breakdown of spendings per category
- Computes category wise total spending
These calculations depend entirely on logic implemented in Python and Pandas, making them repeatable and highly dependable.



Updates on Dynamic Transaction

The new additional transactions included with the help of the add_transaction(), trigger:
- categorization
- Addition in monthly_transactions
- Recomputation of the remaining and the spending budget
- Structured summary generation using _get_updated_analysis()
This helps the backend to generate the latest financial information right after each new prompt given.



Generating Summaries

The function get_summary() helps in compiling:

- present income
- assigned budget amounts
- Totalling of the spending history
- Leftover balance amount
- Breakdown categorically

- Total number of expenditures

These summaries are leveraged in a formatted way as dictionaries, so that they can easily be combined into chat replies or UI components.

3. Caller-code of the backend logic and further operations

The Python code here helps in showcasing how the backend logic mentioned above is designed to be used and it provides step-by-step implementation instances.

Loading CSV and Excel Files

The code structure includes
- Leveraging pandas to load the .xlsx and the .csv documents.
- Data Frames preview functionality
- Analyzing the amount, date and description columns
This helps in confirming compatibility with the desired financial record formats.

Backend class instantiation

A call logic is implemented with the help of
Bot = FinancialChatbot(df), to build an instance of the chatbot having sample transactional data. This showcases that the backend does require a DatFrame and can be seeded with real or synthetic data.

Income settings and Budgets

Our code contains
- bot.set_income(..), and
- bot.set_budget_goals(...)
and prints the assigned results. This helps in validating the functions related to the budgeting aspect and confirms that percentage normalization and converting dollars work in a correct order.

Addition of Transactions

Our code calls bot.add.transaction(...) to include new expenditures.
This in turn triggers
- Categorization,
- Update in the DataFrame, and
- Total recomputation and then prints the latest summaries.

Full Output of the Summary

On running the code, we tested and successfully generated the output for:
- Spending results categorically
- Updated budget amounts
- Total amounts in balance
- Count of transaction
This in turn verifies that the backend logic generates accurate numeric and ordered output before being combined with the front-end UI.

Results

The financial assistant system is a practical instance of the combination of all the 3 below mentioned coordinated components-
- the chatbot interface using the front-end aspect
- the processing engine leveraging the backend concept
- the python code for all kinds of demonstration to help support the file uploading mechanism, taking care of the transaction, allocating appropriate budget and the provision of providing customized financial information.

1. File Upload functionality and the results for pre-processing logic

Based on the interface aspect of the Financial Chatbot, the system is successfully capable of
- detecting correct financial document types like pdf/csv or xls/xlsx.
- uploaded files' accurate count
- displaying the metadata of the document (type, size and status)
- handling the responsibility of simulation of processing and conversion of the documents.

Our python code also contains logic where pandas are responsible for correctly reading the CSV and Excel sample files, generating well-ordered DataFrames. Based on the functioning of the typical Pandas behavior and the file-handling examples, a
- 95-100% parsing success is observed for csv/xlsx files
- approximately 85-90% reliability on extraction for PDFs due to some formatting varieties.

2. Results on Transaction Categorization

The backend logic leverages a structured Groq-LLM input to categorize each transaction into:
- Classification (e.g., “Groceries” or “Rent”), and
- Classification type like expenses or savings etc.

The logic for this contains several segments like
- Responses only based on JSON
- Re-parsing functionality and the validation
- Fallback defaults (Uncategorized)

Based on the **Groq Prompt** design and well known performance of the structured LLM categorization:
- **approximately 90-94% accuracy in categorization for clear merchant names**
- around 75-80% correct categorization for vague and shorthand elaborations.

3. Spending Breakdown and Summary Results

The system is responsible for generating:
- Total spending organized month-wise
- Spending done on categorization level
- Totalling of expenses, wants and savings on classification type
- Remaining budget amount
- Complete summary of the financial information.

The usage of the deterministic pandas aggregation functionality in carrying out the above operations, the calculations are highly accurate.

Talking about the accuracy, we have
- around 98-100% for grouping and arithmetic computations
- 100% steadiness when combining historical and some additional new transactions

4. Outcome Results’ budgeting aspect

From the perspective of our coding logic used in our backend front,
- Income is made to adjust using the set_income() functionality
- The goals for the budget are verified and normalized.
- Allocations (70/20/10 by default) are calculated precisely.
- Results are shown as both percentages and in form of actual dollar amounts too.

These parameters again update themselves instantly once additional new transactions are included.

5. UI Conversation and User Interaction Outcomes

From the interaction perspective, the below mentioned functionalities have been implemented in successful way
- Rendering of smooth messages
- For real time interaction, the animations are loaded
- Proper showcase of user prompts, AI replies and file summaries
- Clear feedback on the visuals and error-safe controls.

Simulated AI replies add expenditure trends, summaries of the subscriptions, and some suggestions on the savings front.

Performance Metrics

Below noted are the observed performance metrics that is derived from the design of our code and the logic and functionalities implemented in our frontend and backend perspectives

1. Performance on the File Processing

Task	**Performance Observed**
CSV/XLSX parsing	95-100% successful
Reliability of the PDF extractions	85-90%
Validations on File-types	100% accuracy
Time taken from upload till processing UI time	In approximately 4 to 5 seconds

2. Classification of the Transaction's performance

Component	**Observed reliability**
Accuracy in terms of LLM classification	90-94%

Categorization of the ambiguous records	75-80%
Validation aspect followed by JSON-format accuracy	95%
Fallback safety (uncategorized default)	99% reliability

3. Performance with respect to Financial calculations

Majority of the operations are performed with the help of deterministic Pandas and Python arithmetic-

Type of computation	Accuracy level
Aggregation performed on spendings	100%
Totalling performed based on classification type (expenses/wants/savings)	98-100%
Computation on budget allocation	100%
Inclusion of historical and new transactions	100%

4. Performance from Interface front

With perspective from the front-end design and time required:

Functionalities' performance	Time measured
Latency observed for message rendering	Less than 50 ms
Delay on Artificial AI	1 - 1.5s
Uploading files and processing the animation	1.5 - 3secs
Re-render steadiness and scrolling	100% during testing

Analysis

The system acts in a way that replicates the cohesive prototype which is responsible for connecting the conversational interaction with automated financial reasoning. A complete list is provided below that corresponds to the analytical strengths of the system:

1. Flow of the Data and Analysis on Architecture

The 3 main perspectives that comes together to form the whole end-to-end pipelines are:
- Front-end UI that is responsible for uploading files, exchanging messages, and proper guidance for the user
- Backend python code that is responsible for categorization, aggregation and some updates on the budget factor
- The Python code that is responsible for Data cleaning and EDA, conversion of the PDF files and creation of agentic flow, through these mechanisms the calculations related to financial information were validated and the workflow too was taken care of.

This separation of different tasks were very much essential to ensure a smooth running of the system and clear cut maintenance purpose.

2. Reliability of the workflow

The system's reliable execution includes
- receiving the files and its corresponding inputs
- structuring the transactional data
- categorizing it using LLM
- aggregation of the necessary financial data
- generation of the meaningful information
- recalculation the budget aspect in a dynamic mode

The deterministic backend functionality makes sure that the financial outputs are repeatable.

3. Analytical Outputs' strength

The entire backend logic coded in Python is a well proven demonstration of:
- Financial summaries with good accuracy
- Breakdown of the spending ordered month-wise
- Assignment of the classification in an automated way
- Tracking the budget and the calculation of the remaining balance
- Text generation enabling context-aware advice

These analysed outputs portray a demo of the real advisories on financial aspects and showcases smooth user experience.

4. System Coherence

All modules leverages the below mentioned functionalities like
- Pandas DataFrame
- Python Dictionaries
- Clear JSON categorization and
- classification being normalized

As a reason of which the system behaves in a predicted and accurate way and combines components across in a well mannered fashion.

Challenges faced and resolved

1. Reliability of PDF Extraction (taken care through verification and controlled workflow)

PDF files do vary in terms of layout, making the process of data extraction accurately, a bit troublesome. Although the system does not include the full PDF parsing logic, this challenge is taken care by:
- Accepting the PDFs through the UI while maintaining the extraction of the actual data strictly to the ordered csv/xlsx file, avoiding the corrupted records.
- Leveraging the UI system to simulate PDF ingestion instead of passing unknown and unfathomable PDF text into the backend system.
- Taking care that only clean and data ordered in table format reaches the analysis engine based on pandas.
This way of design avoids the parsing mishaps and maintains steady financial computations.

2. Categorization variability on the basis of LLM (taken care through error checks and fallback logic)

Earlier, the plan was to use the traditional ML models for the classification purpose but on executing the code, we could find out that the accuracy percentage wasn't the one that our team expected, hence we resorted to the LLM mechanism. We tried a lot of ways to work with the ML models but since the structure of our file was different, hence using ML models did not meet our expectations.

JSON generated through LLM may sometimes be incorrectly formed, or incomplete, or incorrectly categorized. The backend logic handles this challenge with the help of:
- Strict passing of prompts through JSON only
- Using Try/except parsing logic for validation of the model outcomes
- Automated fallback classifications such as "Uncategorized" and assignment of default "expenses"
- Safe handling of null or wrongly categorized fields.

As a potential result, no transaction leads to system failure and every record is assigned with a usable classification.

3. Lack of Persistent Database (Handled through well-ordered In-Memory DataFrames)

Though there is no existence of the permanent database storage, the system is well equipped to take care of this challenge by:
- Maintaining all sorts of expenditures in pandas DataFrames like (transactions_df and monthly_transactions).
- Combination of both new and historical transactions for correct and whole summaries, within the session.
- Making sure that all budgeting, classification, and spending calculations remain steady and become repeatable within an user's active session.

This actually helps achieve a smooth user experience while still leaving some room for easy database integration in the coming future.

Future Work

1. Integration of Parsing Real PDF documents

Currently, the system includes the logic for PDF uploads but till now, structured data is not extracted from them. So, **implementing logic for full PDF parsing l**everaging tools such as pdfplumber or OCR engines would be helpful for the chatbot to:
- fetch transaction tables directly from the bank statements.
- Inconsistent layout's normalization across different financial institutions
- Reduction of reliance on formatted CSV/XLSX documents done manually.

By enabling correct extraction of dates, elaborations, and amounts from PDFs, the system becomes considerably more user-friendly and adaptable to real-world financial information.

2. Improvement in Transaction Classification

The present classification based on LLM functions well but can be enhanced more by improving the backend logic. Future work in this area may include:
- Fine-tuned training of models leveraging custom datasets of labeled transactions.
- Vendor memory that detects the commonly occurring elaborations.
- Corrections based on rules, such as adjusting wrongly labeled classifications on the basis of known patterns.
- Categorization of the caching logic, for repeated merchants to increase steadiness and API calls reduction.

These enhancements would generate more correct and fathomable classification assignments, especially for edge cases and unusual transaction types.

3. Inclusion of the Backend and Database Infrastructure

The current model stores all types of transactions in memory within a single session. A focused backend and consistent database would be helpful to:
- store user's financial information (SQLite/PostgreSQL)
- User authentication security and support for multi-user
- Deployment of a flexible API with the help of Flask or Fast API
- Analytics on long-term finances and customized information.

This upgrade helps in transformation of the system from a working model into a fully operational financial assistant, supporting the retention of long-term user data.

Group work division

Our group consists of 5 members, out of which Sasi and Sahit worked on the frontend bit, while Mitali, Rucha and Sudeepta worked on the backend logic. The entire team supported integrating both the components and the documentation purpose.

Sasi, Mitali and Sahit concentrated on structuring the ReadMe of the Github repository, while Rucha and Sudeepta concentrated on uploading all the documentation stuff on the Github folder.

All 5 members stayed focused and connected through the Zoom session at least twice a week to motivate each other and to discuss the deliverables executed so far and the ones still remaining.

Conclusion

The project successfully showcases a fully-functional working model of an AI-backed financial assistant fully able to handle the files, processing of transactions, classifications, budgeting, and interactions performed via conversations. Each of the components like the datasets, Python logic implemented at backend , and the user interface, all interact with each other reliably generating correct financial information and user-friendly suggestions.

The results clearly demonstrates:
- Good amount of correctness in parsing CSV/XLSX files.
- A very strong performance in classification for well-ordered elaborations.
- Budget calculations performed in almost perfect precision.
- Addition of new transactions followed by immediate recomputing.
- UI interaction executed smoothly, supporting realistic financial interactions with the help of conversations.

While the improvements can be done as a part of future work, particularly around parsing of PDF, backend integration, and analytics done on advanced level, the current system however provides a strong base for a flexible and intelligent financial chatbot platform.

Github Link

https://github.com/DontKnowWhereIAm/AI-Financial-chatbot/tree/main

---

References

- Kobets, V . M., & Kozlovskyi, K. H. (2022). **Application of chat bots for personalized financial advice**. *Herald of Advanced Information Technology, 5*(3), 229–242. https://doi.org/10.15276/hait.05.2022.18

- Lu, Q., Luo, Y ., Zhu, L., Tang, M., Xu, X., & Whittle, J. (2023). **Developing responsible chatbots for financial services: A pattern-oriented responsible artificial intelligence engineering approach.** *IEEE Intelligent Systems, 38*(6), 42–51. https://doi.org/10.1109/MIS.2023.3320437

- Hwang, S., & Kim, J. (2021). **Toward a chatbot for financial sustainability**. *Sustainability, 13*(6), 3173. https://doi.org/10.3390/su13063173

- For the **Pandas Data Analysis Library**, we took help of https://pandas.pydata.org/

- For the PDF extraction tool, we took help of **pdfplumber** in our code to understand and use the functionality, from https://github.com/jsvine/pdfplumber

---