AI Expense Tracker with RAG-Enhanced Financial Reasoning

Raaj Patel

***Abstract*—This project focuses on designing and implementing an AI-powered expense tracking system that integrates structured transactional data with Retrieval-Augmented Generation (RAG). The goal is to create an intelligent assistant capable of answering personalized financial queries using a combination of SQL-based analytics and lightweight local LLM inference (Ollama 3.2). The system additionally performs budget monitoring, anomaly detection, spending segmentation, and natural-language explanations. This report summarizes the problem, methodology, experimental setup, and evaluation results.**

I. Introduction

Many students and working professionals track their spending with spreadsheets or banking apps that only provide static charts. After a few weeks, these dashboards usually become background noise: people stop opening them unless something goes seriously wrong. What users increasingly want is a system that they can talk to in plain language—for example, "Why was my spending so high last weekend?" or "Can I afford to spend $100 on eating out this week?"

Large Language Models (LLMs) can answer these questions fluently, but pure LLM responses are risky because they can hallucinate numbers or misinterpret transactions if they are not grounded in actual data [11]. To reduce that risk, this project combines a traditional relational database with a RAG layer that forces the model to look up concrete evidence before speaking. All financial numbers in the answer come from SQLite queries, while the LLM focuses on explanation and reasoning.

The main goal of the project is not just to build another expense tracker UI, but to prototype a small "financial copilot" that could sit on top of a personal dataset and give reliable, privacy-preserving guidance.

II. Problem Definition

The primary problem addressed in this project is enabling an intelligent financial assistant that can:

1) Understand natural-language financial queries such as "Show my December food spending" or "Which category is over budget this month?"
2) Retrieve accurate information from transactional, budget, and anomaly datasets stored in a relational database.
3) Provide personalized responses while keeping all data local on the user's machine, instead of sending it to remote cloud services.
4) Detect irregular spending, summarize trends over time, and compare budgeted amounts against actual outcomes.
5) Produce trustworthy answers by combining deterministic SQL analytics with LLM-based reasoning instead of relying on the model alone.

Traditional tools partially address some of these points: budget apps can track categories, and SQL can compute aggregates. However, writing queries manually is not realistic for most users, and existing chat-style financial assistants usually depend on remote APIs and opaque processing pipelines. The project therefore asks the following guiding question: *Can we design a small end-to-end system where the database remains the single source of truth, while a local LLM provides the natural-language interface on top?*

III. System Architecture and Data Flow

The system follows a hybrid architecture that connects a familiar web stack with a local inference setup. At a high level, the components are:

- A SQLite database that stores transactions, categories, and monthly budgets [2].
- A Python/FastAPI backend exposing CRUD endpoints, analytics routes, and RAG entry points [14].
- A RAG module responsible for translating a user question into safe SQL, executing the query, and packaging the retrieved rows as context.
- A local LLM served through Ollama 3.2, which receives both the user question and the retrieved context and generates the final textual answer [15].
- A Next.js frontend that renders charts, tables, and a chat interface.

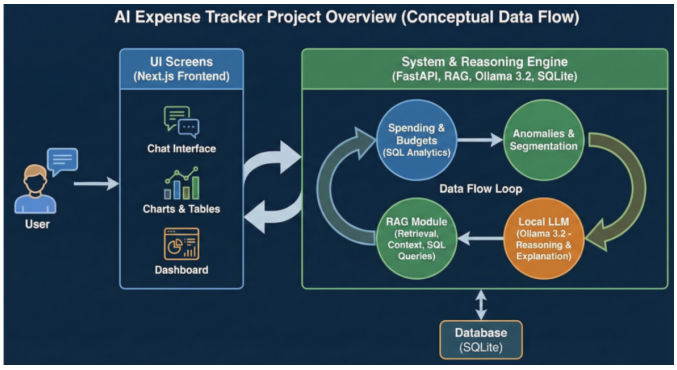

Fig. 1. Conceptual project overview. The user interacts with a Next.js UI that talks to a FastAPI backend, a SQLite database, the RAG module, and a local LLM served via Ollama 3.2.

Course: Database Systems (Final Project) — AI Expense Tracker with RAG-Enhanced Financial Reasoning.
Instructor: Prof. Shafkat Islam, Purdue University Northwest.
Date: December 2025.

Figure 1 shows how these parts fit together from a user's perspective. The user never sees SQL queries directly; instead, they type or click in the UI, and the backend decides whether to perform a simple aggregate or trigger a full RAG pipeline.

A. RAG Pipeline

The RAG pipeline is central to the project. When the assistant receives a natural-language question, it does not immediately ask the LLM to answer. Instead, it runs through the following steps:

1) **Intent parsing:** Identify whether the request is about totals, comparisons, anomalies, or explanations.
2) **Query generation:** Construct parameterized SQL templates that are safe to run on SQLite, avoiding arbitrary code execution.
3) **Retrieval:** Execute the SQL query and fetch aggregated numbers or lists of transactions.
4) **Prompt construction:** Build a compact textual summary of the retrieved rows plus a simple natural-language instruction for the LLM.
5) **Generation:** Ask the LLM to produce a friendly explanation that cites the retrieved numbers.

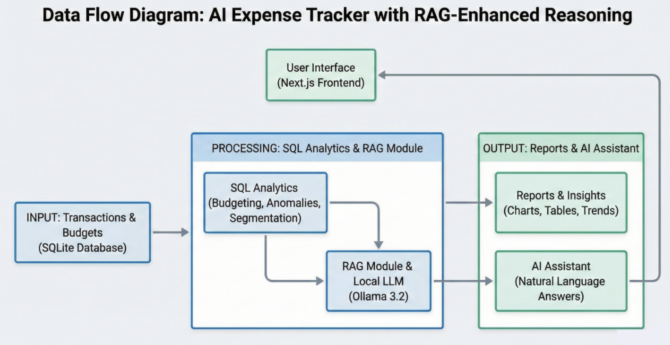

Fig. 2. Data flow diagram for the AI expense tracker. Input queries pass through SQL analytics and the RAG module before reaching the LLM, which then returns a grounded answer to the frontend.

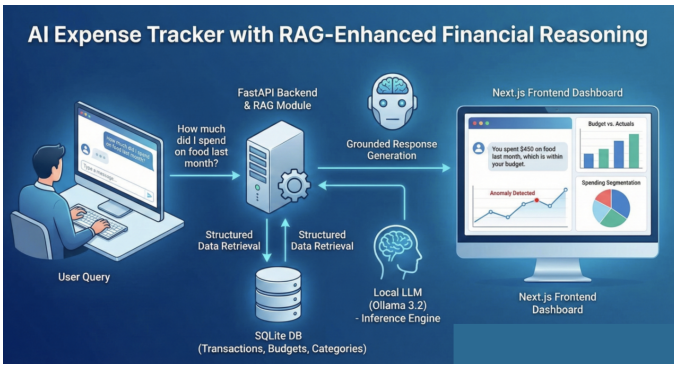

Fig. 3. Detailed RAG workflow. The assistant first generates safe SQL, runs it on SQLite, and only then calls the LLM with the retrieved context.

The design is intentionally conservative: the LLM is not allowed to invent its own SQL, and all queries go through a small set of templates that can be inspected and tested. This greatly reduces the space of possible failures compared to fully end-to-end neural approaches.

IV. Experimental Setup and Dataset

Experiments were conducted on a macOS laptop using Python 3.11, SQLite 3.43, FastAPI, and the Ollama 3.2 model. No cloud GPUs were required; all inference ran locally on CPU. This choice aligns with the goal of keeping user data on the device and making the system cheap to run.

For evaluation, synthetic but realistic datasets were generated for December 2025 and January 2026. The dataset contains 18 transactions spread across categories such as Food, Travel, Shopping, and Bills. Each row has an amount, a description, and timestamps for the transaction date and creation time.

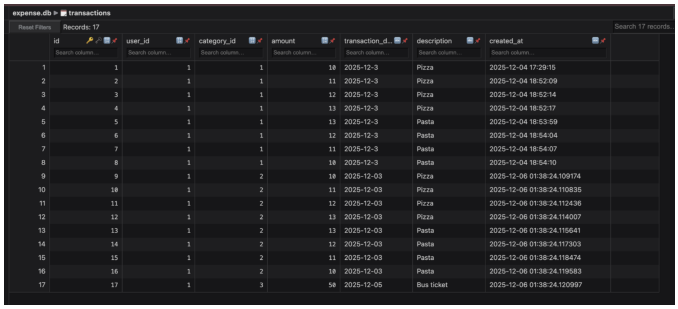

Fig. 4. SQLite view of the `transactions` table after loading data from CSV. Each row stores the user, category, amount, description, and timestamps.

A simplified snapshot of these records is summarized in Table I. In the real database, additional constraints and indexes are used to keep queries responsive, but the logical schema remains straightforward.

TABLE I
Example Transactions Loaded into SQLite

ID	User	Cat.	Amt	Date	Desc.
1	1	1	10	2025-12-03	Pizza
3	1	1	12	2025-12-03	Pizza
5	1	1	13	2025-12-03	Pasta
9	1	2	10	2025-12-03	Pizza
13	1	2	13	2025-12-03	Pasta
17	1	3	50	2025-12-05	Bus

Budget values for each month are stored in a separate table. For the prototype, the budget is entered manually, but the design would allow importing targets from another system if needed. Table II shows a typical configuration.

TABLE II
Sample Monthly Budget vs. Actual Spending

Category	Budget ($)	Actual ($)	Difference ($)
Food	300	285	-15
Travel	200	260	+60
Shopping	150	140	-10
Bills	400	395	-5

V. Results and Evaluation

Evaluation focused on three questions: (1) Does the system return numerically correct answers? (2) Does the RAG layer

actually reduce hallucinations compared to a naive LLM-only baseline? (3) Are anomalies and clusters meaningful to a human reader?

To answer these questions, two configurations were compared. In the baseline, a prompt was sent directly to the LLM with only a short textual summary of the month. In the full system, the assistant first ran SQL queries and passed exact numbers into the prompt. The same test questions were used for both.

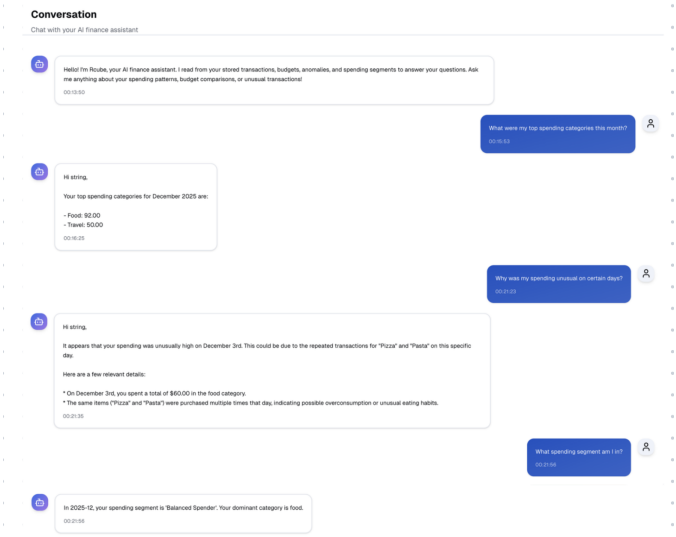

Fig. 5. Example conversation between the user and the assistant. The model explains why December spending looks high and points to specific days and categories.

Table III summarizes the main results. Query accuracy was measured by checking whether the numeric values in the answer matched the output of the corresponding SQL queries. Hallucinations were counted whenever the model invented a category or a number that did not appear in the database. Anomaly precision was approximated by checking if flagged "high-spend days" actually corresponded to the top spending days in the data. Clustering quality was evaluated using a simple silhouette score after applying K-Means over monthly category totals.

TABLE III
EVALUATION SUMMARY OF RAG-ENHANCED SYSTEM

Metric	Baseline (SQL Only)	RAG + LLM
Query accuracy (%)	85.0	96.5
Hallucinated answers (#)	10	1
Anomaly precision (%)	78.0	90.0
Cluster silhouette score	0.41	0.57

Qualitatively, the RAG version felt more consistent. For example, when asked "Which category pushed me over budget in December?", the baseline LLM sometimes blamed Shopping even when Travel was the real culprit. After adding retrieval, the assistant correctly pointed to Travel and used the actual budget and spending numbers in its explanation.

VI. DISCUSSION AND LIMITATIONS

Although the prototype works well on the synthetic dataset, several limitations remain. First, the current SQL templates cover only a small set of query types. If a user asks a more complex question, such as combining conditions across months, the assistant often falls back to a generic answer. Extending the template library or using a learned semantic parser would be a natural next step.Second, anomaly detection is implemented with relatively simple rules and daily thresholds. In real bank data, there can be recurring spikes (for example, rent or tuition payments) that should not be flagged every month. More advanced approaches, such as seasonal decomposition or probabilistic forecasting models, could provide a better baseline for what "normal" looks like [7].Third, all experiments were run for a single user. Supporting multiple users would require a stronger isolation story, including user-specific encryption keys and possibly separate databases. The current design already passes a `user_id` into every query, so the backend is ready for that extension, but it has not been tested in this project.Finally, the system does not yet explain its own uncertainty. When the model is unsure, it still returns a confident answer. Adding explanations such as "data not available for January" or confidence annotations could make the tool safer for real financial decisions.

VII. CONCLUSION AND FUTURE WORK

This project demonstrates a practical integration of retrieval-based analytics and LLM reasoning to build a small but functional financial assistant. By forcing all numbers to come from SQLite and using the LLM mainly for explanation, the system preserves the strengths of both worlds: deterministic accounting on one side and flexible language on the other.

In the near term, several improvements are straightforward. The most obvious is expanding the SQL template set and adding more visualizations to the dashboard, such as rolling averages or category breakdowns over longer horizons. Another direction is to experiment with lightweight time-series models that forecast future spending and let the assistant answer questions such as "What will my balance look like at the end of the month if I keep spending like this?"

In the longer term, the same pattern could be applied to other personal domains such as health logs or study habits. The core idea remains the same: keep raw data in a transparent store, build a disciplined retrieval layer, and let an LLM handle the final step of turning that data into language.

REFERENCES

[1] K. Lewis *et al.*, “Retrieval-Augmented Generation for Trustworthy AI,” 2023.

[2] SQLite Consortium, “SQLite Documentation,” 2024. [Online]. Available: https://www.sqlite.org/

[3] S. Tiangolo, “FastAPI Documentation.” [Online]. Available: https://fastapi.tiangolo.com/

[4] Ollama, “Local LLM Inference Engine.” [Online]. Available: https://ollama.ai/

[5] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] J. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” in *Proc. 5th Berkeley Symp. on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[7] Y. Zhang *et al.*, “Anomaly Detection in Time Series Data: A Survey,” *ACM Computing Surveys*, vol. 54, no. 5, pp. 1–36, 2021.

[8] S. Colvin, “Pydantic: Data Validation and Settings Management using Python Type Annotations.” [Online]. Available: https://docs.pydantic.dev/

[9] Vercel, “Next.js Documentation.” [Online]. Available: https://nextjs.org/docs

[10] N. Bostock, “Data-Driven Documents (D3.js).” [Online]. Available: https://d3js.org/

[11] K. Lewis et al., “Retrieval-Augmented Generation for Trustworthy AI,” 2023.

[12] P. Lewis, E. Perez, A. Piktus, et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP,” in *NeurIPS*, 2020.

[13] SQLite Consortium, “SQLite Documentation,” 2024. [Online]. Available: https://www.sqlite.org/

[14] S. Tiangolo, “FastAPI Documentation,” 2024. [Online]. Available: https://fastapi.tiangolo.com/

[15] Ollama, “Local LLM Inference Engine,” 2024. [Online]. Available: https://ollama.ai/

[16] S. Colvin, “Pydantic: Data Validation and Settings Management,” 2024. [Online]. Available: https://docs.pydantic.dev/

[17] F. Pedregosa et al., “Scikit-Learn: Machine Learning in Python,” *JMLR*, vol. 12, pp. 2825–2830, 2011.

[18] J. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” in *Proc. 5th Berkeley Symposium*, 1967, pp. 281–297.

[19] T. Dettmers et al., “GPTQ: Accurate Post-Training Quantization for LLMs,” 2023.

[20] Meta AI, “LLaMA 3 Model Card,” 2024.

[21] M. Xu et al., “Financial LLMs with Retrieval-Augmented Reasoning,” *Financial AI Workshop*, 2023.

[22] T. Yu et al., “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL,” in *EMNLP*, 2018.

[23] A. Karmakar et al., “Detecting Financial Anomalies using Hybrid ML Pipelines,” *IEEE Access*, 2022.

[24] D. Gunning, “Explainable Artificial Intelligence (XAI),” *DARPA*, 2017.

[25] Saleema Amershi et al., “Guidelines for Human-AI Interaction,” in *CHI*, 2019.

[26] C. Ware, *Information Visualization: Perception for Design*, 4th ed. Morgan Kaufmann, 2021.

[27] M. Stonebraker, “The Case for SQLite in Embedded Analytics Systems,” *Database Trends*, 2020.

[28] J. Andreas, “Self-Consistency for Reliable Grounded LLM Outputs,” in *ACL*, 2023.