Auto-Categorization of Business Transactions

Problem Statement

Design an accounting application that can automatically categorize business transactions, adapting to each customer's unique category preferences. The system should maintain high accuracy even with infrequent user interactions and learn from real-time feedback to improve its performance over time.

Assumptions

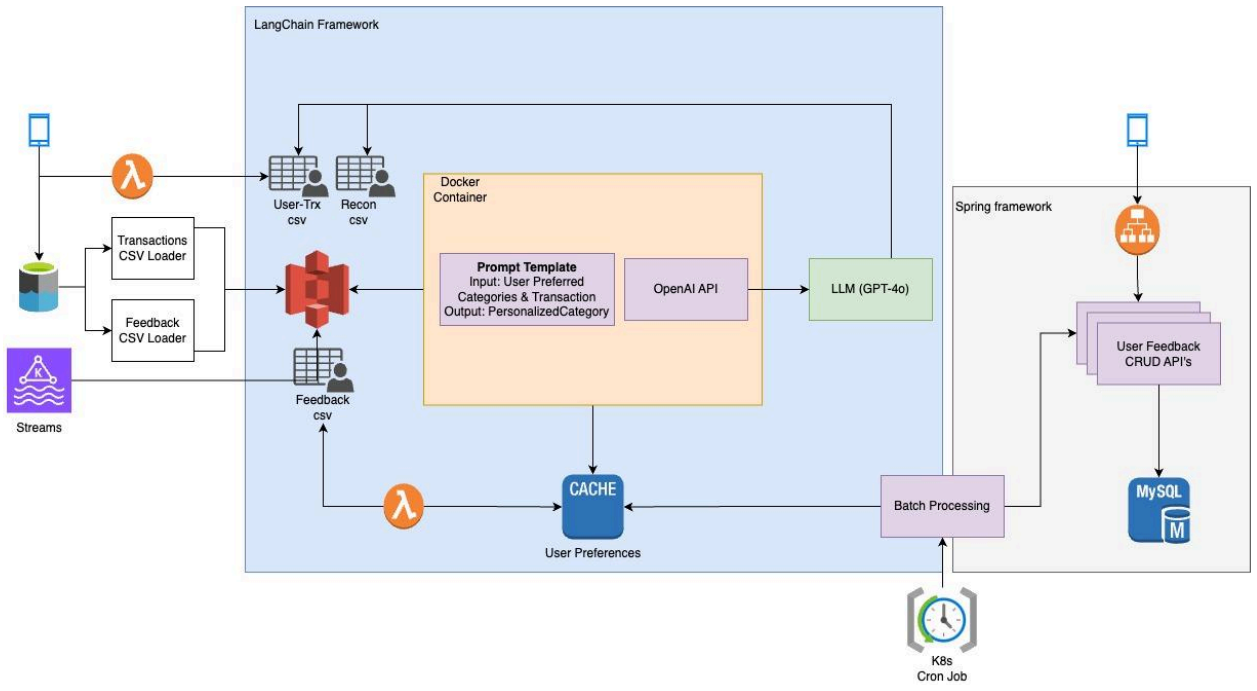
1. **Access to Transaction Data**: All users transaction data is available through batching/ streaming and data is in a structured format for different business (e.g. B2B, B2C, SME).
2. **Generative AI Usage**: The system can utilize Generative AI models on user personal preferences and transaction data.
3. **Historical Data Availability**: Personal preferences, transaction history, and feedback are stored in a database or provided through CSV/API.
4. **Global Category Optimization**: User-segmented Categories are optimized using fine-tuned ML models based on historical feedback and user remarks.

Use Cases

1. **Tax and GST Accounting Decisions**
   Categorized transactions assist in making accurate tax-related decisions, such as determining applicable GST or tax deductions. The system can automatically apply tax rules based on the assigned transaction categories, simplifying accounting processes.
2. **Spending Overview and Analysis**
   Once transactions are categorized, they can be aggregated to provide users with insights into their spending habits. The system generates detailed reports, identifying the largest spending categories and highlighting trends over time.

Algorithms

Algorithm 1: Using LLM's & Prompt Engineering ()



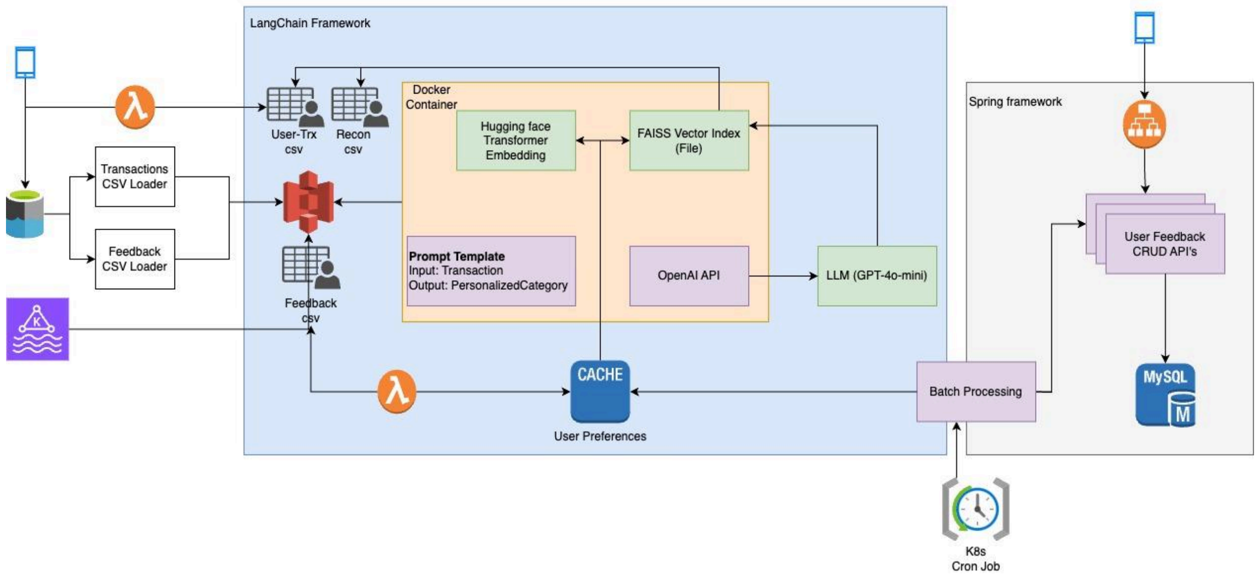
Key Decisions:

- The core decision involves leveraging **Generative AI (LLMs)**, specifically models like GPT-4, to auto-categorize transactions based on user preferences. The LLM is prompted to generate categories for each transaction by taking the transaction description and user-preferred categories into account.
- The **prompt** is carefully engineered to give the LLM clear instructions on how to categorize the transaction. It includes: Transaction details, user's preferred categories.
- Cache-Based User Preference Management for faster LLM interactions and ensures that user-specific data is always accessible.
- Multiple models for cost optimization: The system allows the flexibility to switch between different models, using **LangChain** framework, based on cost and performance requirements. Use **cost-effective models** for low-priority transactions or run **batch processing during off-peak hours** to minimize resource usage.
- **Feedback** Loop: The system aggregates feedback across multiple user transactions and updates the user's preference cache to better reflect the user's evolving categorization preferences. It processes different types of relationships between LLM generated categories and feedback, including 1-to-1, 1-to-many, and many-to-1 relationships. With feedback data being continuously fed into the system, the LLM model adapts over time to improve transaction categorization accuracy. This adaptability is crucial for handling diverse transaction types.

POC Approach: (Github link)

1. Input Data Preparation: CSV files containing distant transaction descriptions are used as input to test the robustness of the category matching. User-specific categories are pre-loaded from a Json **cache** that stores historical preferences for each user.
2. For each transaction, a category is generated by invoking the LLM using the custom **prompt template**. The prompt is designed to take both the transaction description and user-specific preferences as inputs. The LLM responds with the most appropriate category. The categorization results, along with the number of tokens used, are logged, and the total cost is calculated based on the number of tokens consumed.
3. The CSV file containing transactions is updated with the generated personalized categories and saved as an output file.
4. Reconciling Inaccurate Categories: Identify and move transactions that are inaccurately categorized by the LLM to a **reconciliation file** for further review.
5. User feedback is provided on some categorized transactions, indicating corrections or refinements to the categorization. This feedback is collected through various CSV files.

Algorithm 2: Using LLM's and FAISS Vector Index – Category Semantic Search



Key Decisions:

- **Semantic search**: **FAISS** (Facebook AI Similarity Search) is used to index the embeddings of user-preferred categories. This allows for efficient, high-dimensional vector searches, ensuring that the closest match to a user's preferred categories is identified. The **use of vector search** scales better for real-world applications where users may have a large number of transactions and categories.

- **Vector Encoding with SentenceTransformer**: User categories are encoded into vectors using the **Huggingface SentenceTransformer model**. This enables the creation of dense embeddings that FAISS can index for fast and efficient retrieval during the search process.
- **LLM Categorization Followed by Vector Matching:** The LLM (GPT-4o-mini) is used to generate an initial category for each transaction description. The LLM uses a prompt template to produce the category, but this is not directly assigned. Once the LLM generates a category, the system calculates its embedding and performs a semantic search in the user's FAISS index to find the closest match from the user's historical categories. The **Euclidean distance** between the LLM-generated category and the user's historical categories determines the match based on the threshold value of distance. If the distance is below a threshold (set to 1 in the POC), the personalized category is retained, else **fallback for low confidence** by considering LLM category or high end LLM for re-categorization with dynamic prompt.
- **Index for each user**: Each user has their own FAISS index based on their historical categories. This decision allows for personalized searches and ensures that each user's transactions are categorized according to their unique preferences.
- **Feedback Loop**: The FAISS index must be updated as user preferences evolve over time. After receiving user feedback or corrections, the system updates the FAISS index to reflect the latest personalized categories.

POC Approach: (Github link)

1. For each user, the personalized categories from the JSON **cache** are encoded using the Huggingface SentenceTransformer (`all-mpnet-base-v2`). These embeddings are stored in a FAISS index. The FAISS index is stored as a **pickle file** for each user. This enables fast loading and searching in the index.
2. Each transaction is first passed through the **LLM (**`gpt-4o-mini`**)** to generate an initial category. The system uses a **concise prompt template** to guide the LLM in generating a category. It is encoded into a vector using the **embeddings** and **semantic search** is then performed on the user's FAISS index to find the closest match among the preferred categories. The system computes the **Euclidean distance** between the LLM-generated vector and the historical vectors and decides based on distance threshold.
3. The personalized category for each transaction is stored in the **output CSV file**.
4. Reconciling Inaccurate Categories: Identify and move transactions that are inaccurately categorized by the LLM to a **reconciliation file** for further review.
5. User feedback is provided on some categorized transactions, indicating corrections or refinements to the categorization. This feedback is collected through various CSV files.

Algorithm 3: Using LLM's and FAISS Vector Index – Transaction Semantic Search

Key Decisions:

- **Semantic Search at Transaction Level**: Unlike Algorithm 2, which primarily indexed categories, Algorithm 3 indexes **transaction descriptions** combined with the UserId/Username. By embedding both the transaction description and user ID/name, the system ensures that transaction categorization remains personalized to each user.This allows the system to semantically search for similar transaction descriptions across users, providing a personalized experience.
- **Vector Encoding with SentenceTransformer**: User transactions descriptions are encoded into vectors using the **Huggingface SentenceTransformer model**.
- **FAISS First, LLM Second:** Performs a semantic search in the FAISS index to find the closest matching transaction. Retrieves the **personalized category** associated with the matched transaction. If a match is found with a distance below a certain threshold, the historical personalized category is used. If no match is found, or the distance exceeds the threshold, the category is updated based on the LLM-generated category.

Comparison Between Algorithm 1 and Algorithm 2

Criteria	Algorithm 1: LLM and Prompt Engineering	Algorithm 2: LLM and FAISS Vector Index (Category Semantic Search)
Core Approach	Leverages LLMs (GPT-4) with dynamic prompt engg. to generate personalized categories based on user preferences.	Uses LLMs for initial categorization, followed by a semantic search in a FAISS vector for personalized category matching.
Prompt Template	It is used to guide the LLM to generate categories based on a user's preferences.	It is used to guide the LLM to generate categories without user's preferences
Decision Mechanism	LLM outputs are taken as final personalized categories if they align with user preferences	LLM's category is combined with a FAISS vector search, and the closest match (below a Euclidean distance threshold)
User Preferences Management	Cached in Redis, directly referenced during LLM categorization.	Cached both in Redis and FAISS vectors. Updates to user preferences require re-building the FAISS index.
Cost	High cost due to high-end LLM model usage and high token consumption for every user's preferred category.	Token usage is optimized by using cost-effective models and reduced prompt tokens. But additional processing cost associated with the FAISS index.
Performance	LLM processing can be computationally expensive and may have higher latency, especially for larger datasets. Dependent entirely on LLM processing speed & cost. Slightly slower for real-time responses.	Performance is improved as the FAISS vector index allows for faster category matching. FAISS reduces reliance on expensive LLM processing for every transaction.

Accuracy	Relies heavily on the LLM's output and user preferences, which can sometimes result in sub-optimal or generic category selections.	Offers higher accuracy in category matching due to semantic search in FAISS. The vector search ensures more precise and contextually relevant matches
Scaling	Scales well with small to medium datasets, as the LLM does all the heavy lifting. Might become costly or slow with large datasets.	Scales efficiently with large datasets due to FAISS's ability to handle high dimensional vector searches quickly. Indexing overhead exists but is manageable.
Real time learning	Learns from real-time user corrections but requires updates to the LLM's prompts	More sophisticated as the FAISS index is updated with feedback, allowing the system to dynamically adjust to new preferences.
Initial Setup Complexity	Simpler to implement, as it solely relies on LLMs and prompt engineering without requiring additional vector-based infrastructure.	More complex setup due to the need for encoding, maintaining FAISS indices, and keeping track of vector embeddings alongside LLM-generated categories.
Use Cases	Best for systems requiring real-time dynamic generation of categories where personalization is a core focus.	Ideal for scenarios where a balance between performance, cost, and accuracy is required, especially for personalized categories.
Maintenance	Comparatively easier since it involves managing cache & LLM updates. Requires occasional tuning of LLM prompts.	Higher maintenance due to the FAISS index needing updates, re-indexing after feedback, managing both the vector space & the LLM.
Explainability	Category decisions are slightly harder to explain as they rely on LLM "black-box" logic. Can leverage **Explainable AI (XAI)** frameworks.	More explainable as FAISS provides a clear distance-based measure for categories, giving insight into why a particular category was chosen.

Conclusion:

- **Algorithm 1** (LLM and Prompt Engineering) is more suited for smaller datasets, simple setups, and teams looking for ease of use with limited overhead. It relies heavily on LLM outputs, which makes it flexible but less precise in handling large volumes of transactions or complex categories.
- **Algorithm 2** (LLM and FAISS Vector Index) excels with large datasets and complex, semantically similar categories. By leveraging vector-based semantic search, it provides a more scalable and precise solution but comes with higher setup and maintenance costs due to FAISS index handling and embedding management.

Performance Measurement Plan

- Tracks model performance (accuracy of categorizations, error rates, and user satisfaction) using **Cosine Similarity** / **Euclidean Distance**, Mean Reciprocal Rank (**MRR**), Mean Average Precision (**MAP**) etc.
- **Continuous monitoring** for model drift and **embedding drift** in the FAISS index.
- **Precision/Recall:** To measure the accuracy of the auto-categorization, we can calculate precision (correct matches out of total predicted matches) and recall (correct matches out of total actual matches) using similarity thresholds.
- **Latency:** Performance Profiling using PyTorch profiler for **embedding models** like **Huggingface Transformers**, these profilers can help measure the performance of the embedding generation process like Model inference time, GPU utilization **etc.**

Feedback Loop Design

Feedback Loop Interface:

- Provides users with a way to confirm, reject, or modify the suggested transaction categories using CRUD APIs for feedback.
- Gather metadata around user interactions (time spent, frequency of category changes) using Read API and user feedback SQL DB.

Real-Time Learning Module: Models can leverage incremental learning, keeping the model lightweight. System records the feedback and updates the categorization user preferences and Vector Index. **Dual-cache mechanism** where user feedback immediately updates an in-memory cache, which periodically updates the FAISS index in batches. This will ensure that user feedback is reflected in the categorization while minimizing index update costs.

System Architecture Components

Transaction Ingestion Layer:

- **Sources**: Accounting APIs, Streaming, Batching using CSV imports, etc.
- Implements a parser to standardize the format of incoming transactions.
- Real-time ingestion via Kafka for streaming solutions or batching via files.

Categorization Engine (Generative AI-Supported):

- **Batch Categorization**: Leverages LLM for user-preferred category assignment. Combines LLM + Vector Index (FAISS) for personalized category matching.
- **Incremental Categorization**: Same LLM-based process in real-time or batch mode. Semantic search via FAISS for enhanced personalization.

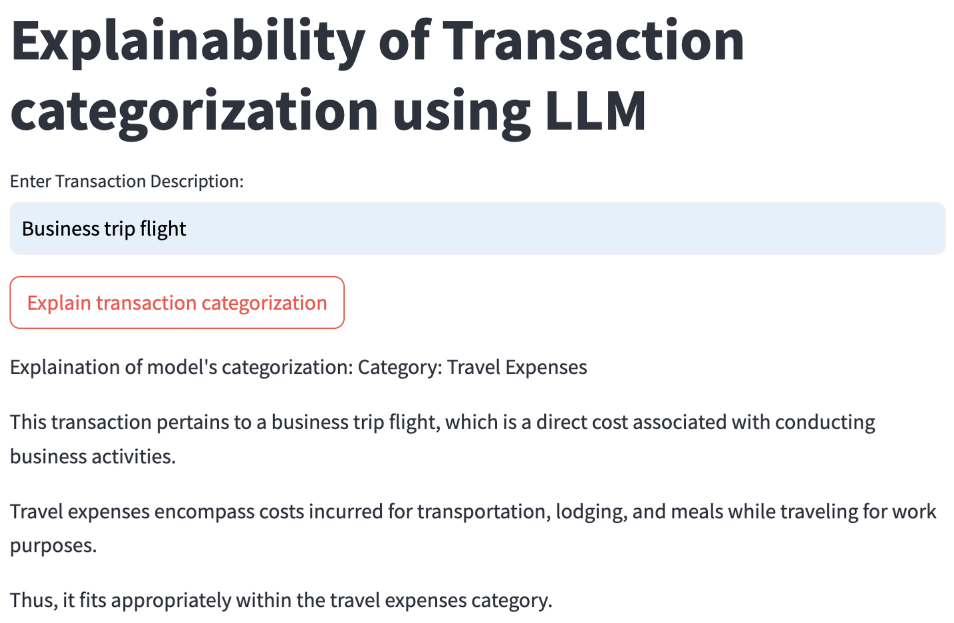
- **Feedback Loop**: If a user changes a suggested category, feedback is processed in real-time or batch mode. Using **Few-Shot Learning** to personalize categories even with minimal user data.

User Profile and Feedback Repository:

- **Profile Storage**: Key-Value Store for user preferences. Vector DB for feedback-based personalization using FAISS.
- **Feedback Storage**: SQL DB and Redis LFU cache to track feedback and preference updates efficiently.

Future Scope

Explainability



Explainability of Transaction categorization using LLM

Enter Transaction Description:

Business trip flight

Explain transaction categorization

Explaination of model's categorization: Category: Travel Expenses

This transaction pertains to a business trip flight, which is a direct cost associated with conducting business activities.

Travel expenses encompass costs incurred for transportation, lodging, and meals while traveling for work purposes.

Thus, it fits appropriately within the travel expenses category.

Error Handling

- **Fallback Mechanism**: If a transaction can’t be categorized, it defaults to “Uncategorized” and requests user input. Or, Instead of relying solely on user input, the system can attempt switching to a higher-powered LLM model for more accurate categorization or a manual rule-based system.
- **Confidence Threshold**: When the model is uncertain, the system notifies the user
- **Feedback system** with **Event-based** (e.g., Kafka) for real-time feedback capture.
- Reinforcement Learning for real-time learning and adapting based on user corrections.