AI Financial Advisor: AI-Driven Hyper-Personalization & Recommendation

Contents

1. Overview

1.1. Purpose

The **AI-Driven Hyper-Personalization & Recommendations** system is designed to enhance user experiences by leveraging **Generative AI** and **Retrieval-Augmented Generation (RAG)** techniques. It provides **personalized recommendations** by analysing diverse data sources such as **customer profiles, purchase history, social media activity, sentiment data, and demographics**.

This solution automates **hyper-personalization** by:

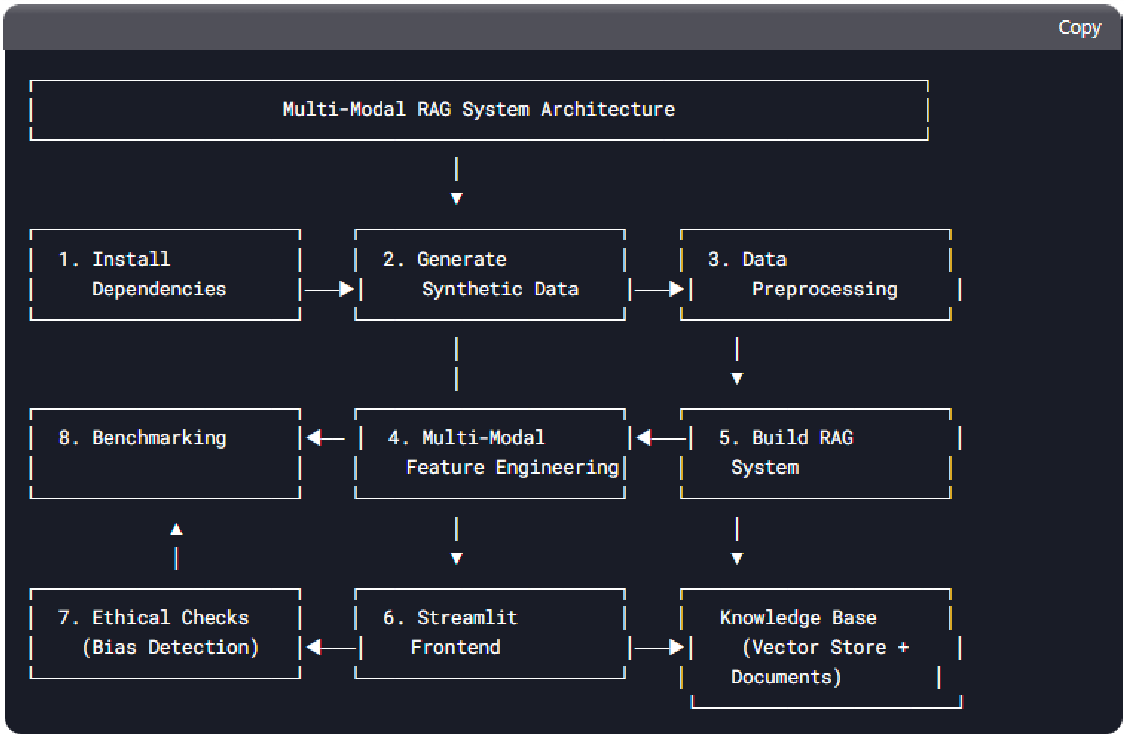
- Generating **targeted recommendations** in real-time
- Incorporating **structured & unstructured data** using **multi-modal learning**
- Ensuring **fair and unbiased AI-driven suggestions** through **ethical checks**
- Using an **adaptive feedback loop** to improve recommendations over time

By integrating **Large Language Models (LLMs)** and **vector-based retrieval**, the system aims to **bridge the gap between user intent and content delivery**, ultimately **boosting engagement, conversions, and customer satisfaction** for businesses.

1.2. Key Components

- **Dependency Setup:** Python environment, library installations (PyTorch, Transformers, LangChain, etc.), and API key configuration.
- **Synthetic Data Generation:** Creating realistic structured data (and pairing with unstructured data using Multi-Modal pairing).
- **Data Preprocessing:** Cleaning, normalization, and train/test split.
- **Multi-Modal Feature Engineering:** Generating text embeddings (using models like BERT/MPNet).
- **RAG System Construction:** Setting up a vector database (using FAISS) and integrating retrieval with LLM (Mistral/GPT-2).
- **User Interface:** A Streamlit-based frontend for user input and response visualization.
- **Ethical Checks:** Bias detection pipeline and output validation.
- **Benchmarking:** Comparative analysis against baseline models to ensure quality.

2. Architecture Flow



2.1. Detailed Component Flow:

2.1.1. Install Dependencies

- Python environment setup
- Library installation (PyTorch, Transformers, LangChain, etc.)
- API key configurations

2.1.2. Generate Synthetic Data

- Structured data generation (Synthetic databases)
- Multi-modal pairing (text+image combinations)

2.1.3. Data Preprocessing

- Text cleaning/normalization
- Train/test splits

2.1.4. Multi-Modal Feature Engineering

- Text embeddings (BERT/MPNet)

2.1.5.Build RAG System

- Vector database setup (FAISS)
- Retrieval & Generator model (LLM, Mistral)

2.1.6.Streamlit Frontend

- User input interface
- Response visualization

2.1.7.Ethical Checks

- Bias detection pipeline
- Output validation

2.1.8.Benchmarking

- Comparative analysis (vs baselines)

2.2. Data Flow:

- Synthetic data → Preprocessing → Feature Engineering → Vector DB
- User query → Frontend → RAG System → (Retrieval + Generation)
- Output → Frontend + Ethical Checks → User
- System performance → Benchmarking → Feedback loop to improve components

3. Pre-Deployment Setup

3.1. Hardware Requirements

- **Development:**
  - 16GB RAM, 4-core CPU (GPU recommended).

3.2. Software Dependencies

- **Tech Stack**
  - Programming Language: Python
  - Frameworks & Libraries: Synthetic Data Generation: SDV (CTGANSynthesizer)
  - Feature Engineering: Pandas, NumPy, NLTK, Transformers, Huggingface, sentence transformers (distilbert-base-uncased, all- mpnet-base-v2)
  - RAG-Based Recommendation: FAISS, LangChain, GPT-2, Mistral
  - Frontend: Streamlit
  - Bias Detection: AI Fairness 360
  - Benchmarking

- Install required Python packages:

Step 1: Install Dependencies

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
from sdv.metadata import SingleTableMetadata
from sdv.single_table import CTGANSynthesizer
from transformers import BertTokenizer, BertModel
import torch
from tqdm import tqdm
from huggingface_hub import notebook_login
#notebook_login()
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk
nltk.download('vader_lexicon')
import faiss
import numpy as np
from langchain.chains import RetrievalQA
from langchain.llms import BaseLLM
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
from langchain.docstore import InMemoryDocstore
from transformers import pipeline
from langchain_huggingface import HuggingFacePipeline
```

Show hidden output

```
#!pip install pandas numpy sdv transformers faiss-cpu langchain streamlit scikit-learn nltk
#!pip install langchain-community
#!pip install --upgrade langchain-core langchain-community langchain-experimental sentence-transformers
!pip install -U langchain-huggingface
```

3.3. Data Preparation

Sample Data Schema:

```
{
  "customer_id": "string",
  "age": "integer",
  "gender": "categorical",
  "purchase_history": "text",
  "social_media_posts": "text",
  "sentiment_score": "float"
}
```

3.4. Synthetic Data Generation:

Step 2: Generate Synthetic Data

```
[ ]
# --- Customer Profiles ---
customer_metadata = SingleTableMetadata()
customer_metadata.add_column('customer_id', sdtype='id')
customer_metadata.add_column('age', sdtype='numerical')
customer_metadata.add_column('gender', sdtype='categorical')
customer_metadata.add_column('location', sdtype='categorical')
customer_metadata.add_column('interests', sdtype='categorical')
customer_metadata.add_column('income', sdtype='numerical')
customer_metadata.add_column('education', sdtype='categorical')
customer_metadata.add_column('occupation', sdtype='categorical')

# Training data with sample values
customer_data = pd.DataFrame([{
    'customer_id': 1234,
    'age': 45,
    'gender': 'Male',
    'location': 'New York',
    'interests': 'Luxury Shopping and Travel',
    'income': 75000,
    'education': 'MBA',
    'occupation': 'Financial Advisor'
},{
    'customer_id': 1235,
    'age': 32,
    'gender': 'Female',
    'location': 'San Francisco',
    'interests': 'Tech Gadgets',
    'income': 125000,
    'education': 'Masters',
    'occupation': 'Engineer'
}])
```

4. Operational Procedures

4.1. Data Processing Pipeline

Structured Data & Unstructured Data:
Clean and transform structured and unstructured data:

```python
# ---------------------------------------------
# Step3: Multi-modal Data Loading & Processing
# ---------------------------------------------
@st.cache_data
def load_data():
    # Load data with customer_id as string
    customer_df = pd.read_csv("/content/drive/MyDrive/Hackathon2025/customer_profiles.csv")
    customer_df['customer_id'] = customer_df['customer_id'].astype(str)

    social_df = pd.read_csv("/content/drive/MyDrive/Hackathon2025/social_media.csv")
    social_df['customer_id'] = social_df['customer_id'].astype(str)

    transactions_df = pd.read_csv("/content/drive/MyDrive/Hackathon2025/transactions.csv")
    transactions_df['customer_id'] = transactions_df['customer_id'].astype(str)

    # Data processing steps (remain the same)
    social_agg = social_df.groupby('customer_id').agg(
        sentiment_score=('sentiment_score', 'mean'),
        content=('content', lambda x: ' '.join(x.astype(str))))

    transaction_agg = transactions_df.groupby('customer_id').agg(
        avg_spend=('amount', 'mean'),
        total_spend=('amount', 'sum'),
        fav_category=('category', lambda x: x.mode()[0]))

    merged_df = pd.merge(customer_df, social_agg, on='customer_id', how='left')
    merged_df = pd.merge(merged_df, transaction_agg, on='customer_id', how='left')
    merged_df['content'] = merged_df['content'].fillna('')

    # Embedding generation
    model = SentenceTransformer('sentence-transformers/all-mpnet-base-v2')
    merged_df['embedding'] = model.encode(
        merged_df['content'].tolist(),
        batch_size=128,
        convert_to_numpy=True
    ).tolist()
```

4.2. Building the RAG/Recommendation System

4.2.1.Train Model

Retrieve similar data and generate recommendations using LLM(Mistral):

```python
# ------------------------
# AI Recommendation System
# ------------------------

# -----------------------------------------------------
# Step4: load the model with fine-tuned hyper-parameters
# -----------------------------------------------------

@st.cache_resource
def load_llm():
    # Same LLM loading code
    quantization_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_compute_dtype=torch.float16,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_use_double_quant=True)

    model = AutoModelForCausalLM.from_pretrained(
        "mistralai/Mistral-7B-Instruct-v0.2",
        device_map="auto",
        quantization_config=quantization_config,
        torch_dtype=torch.float16)

    tokenizer = AutoTokenizer.from_pretrained(
        "mistralai/Mistral-7B-Instruct-v0.2",
        padding_side="left")
    tokenizer.pad_token = tokenizer.eos_token

    return pipeline(
        "text-generation",
        model=model,
        tokenizer=tokenizer,
        device_map="auto",
        max_new_tokens=256,
        temperature=0.3)

# -----------------------------------------------------
```

4.2.2. Generate Recommendations:

```
# ---------------------------------------------------
# Step5: generate recommendations for a customer record
# ---------------------------------------------------

def generate_recommendation(_pipe, customer_data):
    # Same prompt template
    prompt = f"""<s>[INST] As a financial advisor, analyze:
    - Age: {customer_data['age']}
    - Income: ${customer_data['income']}
    - Recent Spend: ${customer_data['avg_spend']}
    - Interests: {customer_data['interests']}
    - Social Sentiment: {customer_data['sentiment_score']:.2f}

    Recommend 3 financial products and business strategies. Be concise. [/INST]"""

    response = _pipe(
        prompt,
        num_return_sequences=1,
        repetition_penalty=1.2)[0]['generated_text']

    return response.split("[/INST]")[-1].strip()
```

4.3. Ethical Monitoring and Benchmarking

4.3.1.Bias Detection:

Run bias checks using Fairness 360:

- **<u>Gender bias in recommendations:</u>**

Converts gender categories to numbers: Female=0, Male=1, Other=excluded Checks if recommendations/predictions favor one gender over another Uses Disparate Impact Ratio: Formula: (% Female customers getting good recommendations) / (% Male customers getting good recommendations) Perfect Score = 1.0 Example: 0.8 means women get 20% fewer good recommendations

- **<u>Income Fairness Check:</u>**

Splits customers into two income groups: Lower 25% income = 0 Upper 75% income = 1 Measures Statistical Parity Difference: Formula: (% Lower-income good recommendations) - (% Higher-income good recommendations) Perfect Score = 0.0 Example: -0.15 means lower-income group gets 15% fewer good recommendations

```python
# ---------------------
# Step6: Ethical Checks
# ---------------------

@st.cache_data
def check_bias(df):
    # Same bias checking code
    df = df.copy()
    df['gender'] = df['gender'].map({'Female': 0, 'Male': 1, 'Other': -1})
    df = df[df['gender'] != -1]

    try:
        df['income_bin'] = pd.qcut(df['income'], q=[0, 0.25, 1.0], labels=[0, 1]).astype(int)
    except ValueError:
        df['income_bin'] = (df['income'] > df['income'].median()).astype(int)

    np.random.seed(42)
    df['prediction'] = np.random.randint(0, 2, size=len(df))

    dataset = BinaryLabelDataset(
        df=df[['gender', 'income_bin', 'prediction']],
        label_names=['prediction'],
        protected_attribute_names=['gender', 'income_bin'])

    metrics = {}
    gender_counts = df['gender'].value_counts()
    if 0 in gender_counts and 1 in gender_counts:
        metrics['gender_impact'] = ClassificationMetric(
            dataset, dataset,
            unprivileged_groups=[{'gender': 0}],
            privileged_groups=[{'gender': 1}]).disparate_impact()
    else:
        metrics['gender_impact'] = np.nan

    income_counts = df['income_bin'].value_counts()
    if 0 in income_counts and 1 in income_counts:
        metrics['income_fairness'] = ClassificationMetric(
```

4.3.2.Benchmarking:

Perform comparative analysis to ensure the system meets performance metrics.

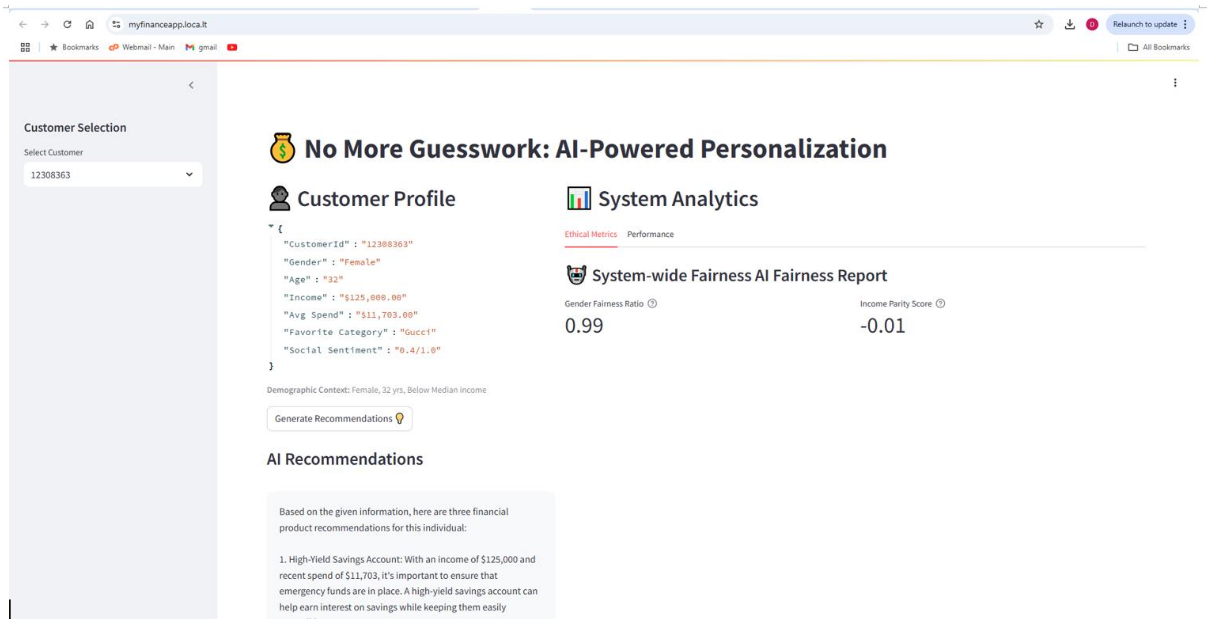
- **<u>Tests system accuracy using:</u>**

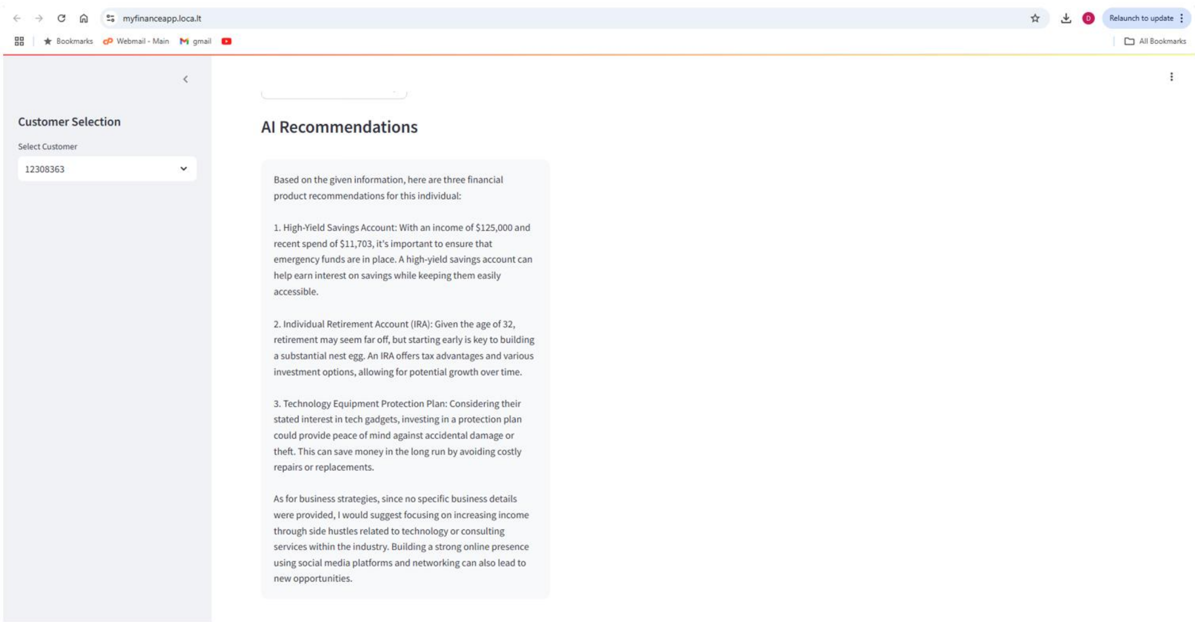
Spending predictions vs actuals (MAE score) Recommendation quality (RMSE score) Pattern recognition capability

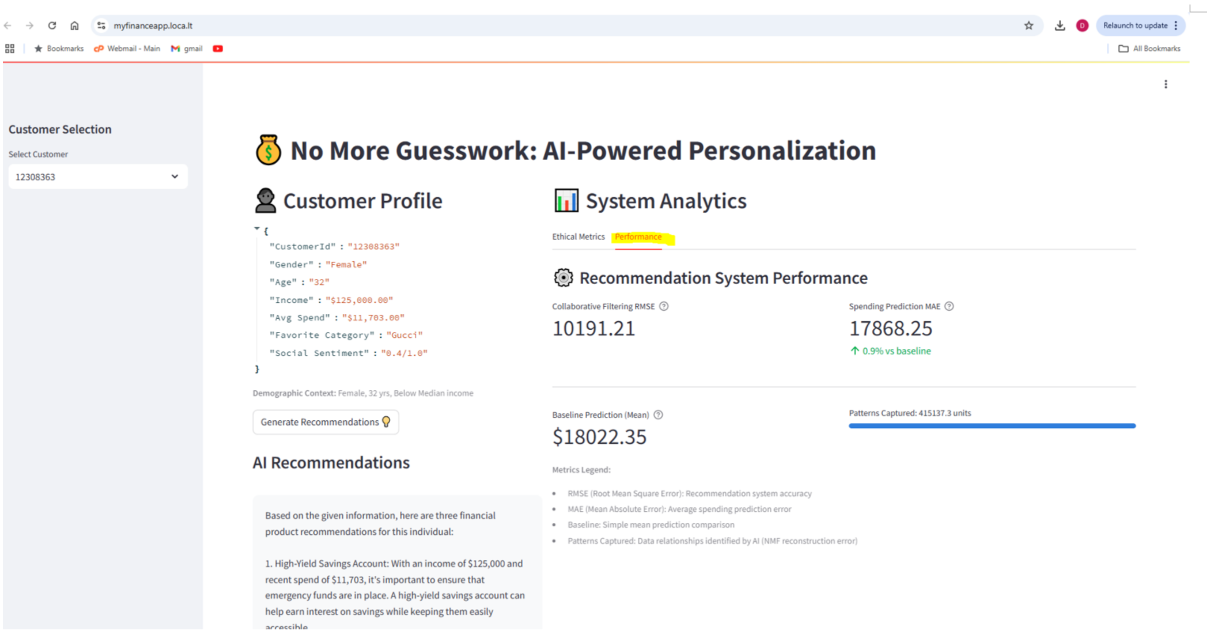
```
# -------------------
# Step7: Benchmarking
# -------------------

@st.cache_data
def run_benchmarking(df):
    """Improved benchmarking with error handling"""
    results = {
        'cf_rmse': np.nan,
        'xgb_mae': np.nan,
        'baseline_mae': np.nan,
        'improvement_pct': np.nan,
        'variance_explained': np.nan
    }

    try:
        # 1. Collaborative Filtering Evaluation
        median_spend = df['avg_spend'].median()
        user_item = df.pivot_table(
            index='customer_id',
            columns='fav_category',
            values='avg_spend'
        ).fillna(median_spend)

        # Train-test split
        train_mask = np.random.rand(len(user_item)) < 0.8
        train = user_item[train_mask]
        test = user_item[~train_mask]

        # NMF modeling
        model = NMF(n_components=min(10, len(train.columns)-1), init='nndsvda')
        W_train = model.fit_transform(train)
        W_test = model.transform(test)
        reconstructed = np.dot(W_test, model.components_)

        results['cf_rmse'] = np.sqrt(mean_squared_error(test.values, reconstructed))
        results['variance_explained'] = model.reconstruction_err_  # Correct attribute
```

5. User Interface (Streamlit)

Interactive Dashboard:
Allows users to input queries and visualize responses:







6. Future Scope

- The benchmarking results can be optimized using different scaling methods during EDA and handling univariate and multi-variate outlier detection techniques and increasing the volume of dataset.
- Add RLHF (Reinforcement Learning from Human Feedback) for fine-tuning.
- Model monitoring dashboard.
- Auto-scaling infrastructure components
- API endpoints for system integration
- Extend to multi-modal outputs (e.g., generate images with Stable Diffusion).

7. Contacts & Support

- Mahesh Gain - GitHub | LinkedIn
- Dhanalakshmi Rajapandiyan - GitHub | LinkedIn
- Ravali M Verghese - GitHub | LinkedIn
- Smita Singh - GitHub | LinkedIn
- Sathyapriya Varadharaj - GitHub | LinkedIn