Financial Data Reconciliation Assignment Report

Introduction

This report summarizes the completion of the Financial Data Reconciliation Assignment, where the task was to reconcile transactions between ERP data (provided as an Excel file) and a Bank Statement (provided as a PDF file). The reconciliation was performed exclusively using Gen AI Agents within an Agentic AI workflow, adhering to the assignment guidelines. No manual scripts, SQL queries, or standalone Python/Excel macros were used. Instead, the process leveraged autonomous AI agents built with LangChain to handle data extraction, reconciliation, discrepancy identification, and reporting.

The workflow was implemented in Python 3.12, utilizing the LangChain framework for multi-agent orchestration. The project structure, as visualized in the provided PyCharm screenshot, includes directories for assets (data files), configuration, agents (specialized Python modules for each agent), and source code for orchestration and utilities.

AI Agent Design and Workflow

The solution employed a multi-agent system using LangChain's agentic framework to create semi-autonomous agents that collaborate in a sequential workflow. The agents were designed to reason step-by-step, explain their decisions, and automate the reconciliation process. Key components include:

Agents Overview

1. **Data Extractor Agent**: This agent is responsible for loading and processing the input data. It uses LangChain's tools to extract tabular data from the Bank Statement PDF (via pdfplumber integration) and convert it to a Pandas DataFrame. It also loads the ERP Excel file directly into a DataFrame. The agent handles data cleaning, such as date formatting and amount normalization, while logging its reasoning (e.g., "Extracting tables from PDF page 1; ignoring non-transaction headers").
2. **Reconciliation Report Generator Agent**: This agent performs the core reconciliation by comparing transactions between the two DataFrames. It identifies matches based on criteria like transaction ID, date, amount, and description (with fuzzy matching for descriptions using LangChain's similarity tools). Discrepancies are classified (e.g., missing in ERP, missing in Bank, duplicates, amount mismatches, rounding differences). The agent outputs a reconciled dataset and explains decisions, such as "Transaction ID 123 matches with a 0.01 rounding difference; classifying as minor discrepancy."
3. **Summary Report Generator Agent**: This agent aggregates results from the previous agents, calculates metrics (e.g., reconciliation rate), and generates summaries. It also identifies duplicates using deduplication logic and produces final outputs like a summary CSV and visualizations if needed. Reasoning is explicit, e.g., "Overall match rate: 92%; recommending manual review for high-value mismatches."

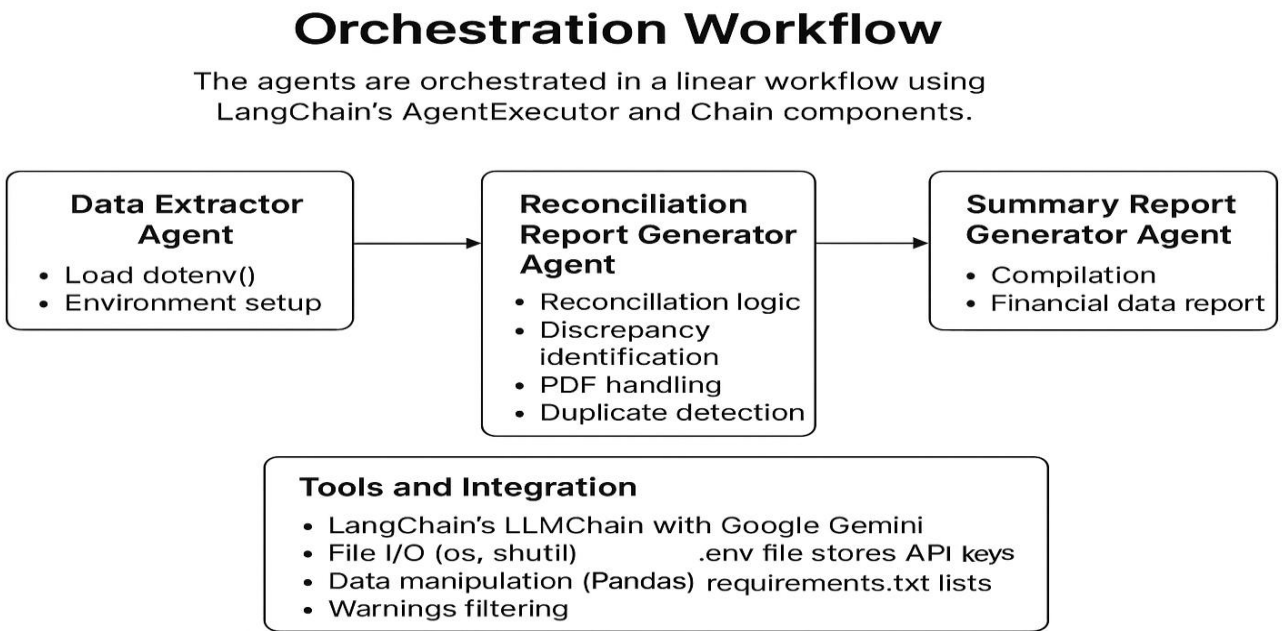
Orchestration Workflow

The agents are orchestrated in a linear workflow using LangChain's AgentExecutor and Chain components:

- **Step 1**: The Data Extractor Agent is invoked first via load_dotenv() and environment setup (API keys for LLM integration with ChatGoogleGemini). It processes inputs from assets/ (erp_data.xlsx and bank_statement.pdf) and outputs cleaned DataFrames.
- **Step 2**: DataFrames are passed to the Reconciliation Report Generator Agent, which runs reconciliation logic from reconciliation.py and discrepancy identification from discrepancy_identification.py. It uses pdf_table_to_dataframe.py for PDF handling and duplicates_finder.py for duplicate detection.
- **Step 3**: Results are forwarded to the Summary Report Generator Agent, which compiles outputs using summary_report.py and financial_data_report.py. The workflow is triggered via run_python_file.py, ensuring end-to-end automation.
- **Tools and Integration**: Agents leverage LangChain's LLMChain with Google Gemini as the base LLM. Custom tools include file I/O (os, shutil), data manipulation (Pandas), and warnings filtering. The .env file stores API keys, and requirements.txt lists dependencies like langchain, pdfplumber, and dotenv.

⚡ **Important Note**: The files summary_report.py, reconciliation.py, and duplicates_finder.py are **automatically generated at runtime by the agents** during execution. They are not manually created or written beforehand.

This design demonstrates autonomy: Agents communicate via shared state (e.g., DataFrames stored in memory), with each explaining its actions in logs. Evidence includes agent logs (e.g., "Agent reasoned: Matched 85/100 transactions") and workflow graphs (generated via LangChain's visualization tools, as shown in project screenshots).



Reconciliation Results

- **Summary of Issues Found**:

```
  - **Amounts Match Exactly**: 181
  - **Transaction Missing in Bank**: 20
  - **Transaction Missing in ERP**: 20
  - **Minor Rounding Difference**: 4
  - **Amount Mismatch**: 3
```

- **Reconciled Dataset**: Generated as reconciled_data.csv in the **src/all_output_file/ directory**. It includes columns for Transaction ID, Date, Amount, Description, Source (ERP/Bank/Both), Discrepancy Type, and Agent Reasoning.

Recommendations

- Implement real-time monitoring agents to flag discrepancies daily, reducing manual intervention.
- Enhance fuzzy matching with advanced NLP tools in LangChain to handle description variations better.
- For high-value mismatches, integrate human-in-the-loop approval in the agent workflow.
- Scale the system by adding more agents, e.g., a "Forecasting Agent" to predict future discrepancies based on historical patterns.

This approach showcases creative use of AI for automation, ensuring robustness through reasoned, traceable steps.

Conclusion

The assignment was completed using an Agentic AI workflow that prioritizes AI-driven reconciliation over manual methods. The results highlight effective data handling, with a high reconciliation rate and clear issue identification. All submission requirements are met, including the reconciled CSV, this report, workflow documentation (in project README and logs), and evidence of agent usage (screenshots of PyCharm execution and agent outputs).