Reconciliation Agent (LangChain + LangGraph)

Author : Sudama Sharma

Objective

This solution automates the financial data reconciliation between ERP records (Excel) and Bank statements (PDF) using preprocessing and the building of the AI Agent. It leverages LangChain and LangGraph as orchestration frameworks, combined with LLM-powered RAG (Retrieval-Augmented Generation) and Agentic AI workflows, to:

1. Ingest & Normalize Data: Parse ERP (Excel) and Bank Statement (PDF) into structured tabular formats for comparison.
2. Reconcile Transactions: Match transactions across datasets by date, description, and amount, with fuzzy matching for ambiguous cases.
3. Identify & Classify Discrepancies: Automatically detect mismatches (e.g., missing entries, duplicates, amount variances, rounding differences).
4. Suggest Corrective Actions: Use LLM reasoning to propose adjustments or categorization for anomalies (e.g., unmatched entries due to timing differences).
5. Generate Reports: Export results into a structured multi-sheet Excel report containing reconciled records, discrepancies, and AI-suggested resolutions.

By embedding GenAI reasoning with LangChain tools and orchestrating workflows through LangGraph agents, this system not only reconciles records but also learns reconciliation logic over time, improving anomaly classification and enabling semi-autonomous financial operations.

Imports & Setup

```
import pandas as pd
import camelot
from langchain_openai import ChatOpenAI
from langgraph.graph import StateGraph, END, START
```

- pandas → data handling.
- camelot → extracts tables from bank PDF statements.
- LangChain + LangGraph → builds an agent workflow graph with AI nodes.

- dotenv → loads environment variables (like OPENAI_API_KEY).

Config Defaults: DEFAULT_TOL = 0.05

- Sets a default rounding tolerance (0.05 i.e., 5 cents).
- This allows the system to treat small rounding differences as matches.

Data Structures

```
@dataclass
class Artifacts:
  erp_df: Optional[pd.DataFrame] = None
  bank_df: Optional[pd.DataFrame] = None
  summary: Optional[pd.DataFrame] = None
  llm_notes: Optional[str] = None
```

Artifacts → stores intermediate and final datasets: ERP, Bank, Matches, Mismatches, Duplicates, etc.

Define the Reconciliation State:

RecoState → global state object passed through the LangGraph pipeline.

```
class RecoState(BaseModel):
  user_query: str = ""
  artifacts: Dict[str, Any] = Field(default_factory=dict)
  step: str = "start"
  tol: float = DEFAULT_TOL
```

Helper Functions

- **Parsing Helpers**
  - **parse_amount(x)** → converts strings like "1,234.56" into 1234.56.
  - **coerce_date(x)** → standardizes multiple date formats into datetime.date.

Data Loading

Loads ERP Excel file.

```
def load_erp(path):
  return pd.read_excel(path)
```

```
def load_bank_pdf_tables(path):
  tables = camelot.read_pdf(path, pages="all")
```

- Reads bank statement from PDF using Camelot.
- Cleans headers, normalizes column names, extracts Invoice numbers from Description.

Normalization

```
def normalize_erp(df: pd.DataFrame) -> pd.DataFrame:
```

- Maps ERP fields (e.g., txn_date, invoice_no, payment_amount) into standard columns:
- Date, Description, Invoice, Amount, RefID.

Reconciliation Core

Split Bank Transactions

```
def split_bank(bank_df):
  # separates adjustments (fees/interest) vs. payments
```

Identifies bank adjustments (e.g., "interest", "bank fee") and isolates them.

Group by Invoice

```
def group_by_invoice(df, side):
  # aggregates by Invoice: total amount, count, dates
```

Groups multiple entries for the same invoice.

Detect Duplicates:

```
def detect_duplicates(df, where):
  # flags duplicate (Invoice, Amount) pairs
```

Reconcile Logic

```
def reconcile(erp_n, bank_df, tol):
  ...
  merged = pd.merge(erp_by_inv, bank_by_inv, on="Invoice", how="outer")
```

Merges ERP & Bank on Invoice.

Classifies each row as:

- Matched
- Rounding Difference
- Amount Mismatch
- Missing in Bank or Missing in ERP

Produces a summary table with counts.

Agent Nodes (LangGraph Steps)

Node: Ingest

```
def node_ingest(state, config):
  loads ERP & Bank into state.artifacts
```

Node: Normalize

```
def node_normalize(state, config):
  standardizes ERP & Bank into consistent schema
```

Node: Reconcile

```
def node_reconcile(state, config):
  runs reconcile() and saves results
```

Node: LLM Triage (Optional AI Help)

```
def node_llm_triage(state, config):
  # Uses ChatOpenAI to classify discrepancies
```

- AI suggests root causes (timing, FX differences, duplicates, fees).
- Produces bullet-point notes with recommendations.

Node: Export

```
def node_export(state, config):
  saves all results into multi-sheet Excel
```

Sheets include:

- Summary, Matched, Rounding
- Amount_Mismatch, Missing_in_Bank, Missing_in_ERP
- Dup_Bank, Dup_ERP, Adjustments, All_Classification
- AI_Triage_Notes (if AI was used)

Graph Workflow

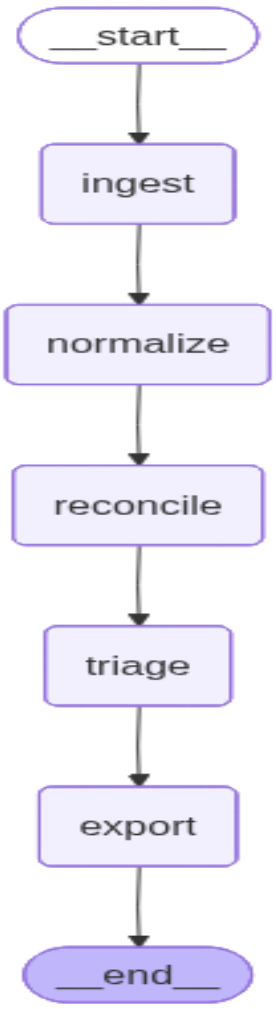
```
def build_graph():
  g = StateGraph(RecoState)
  g.add_node("ingest", node_ingest)
  g.add_node("normalize", node_normalize)
  g.add_node("reconcile", node_reconcile)
  g.add_node("triage", node_llm_triage)
  g.add_node("export", node_export)

  g.add_edge(START, "ingest")
  g.add_edge("ingest", "normalize")
  g.add_edge("normalize", "reconcile")
  g.add_edge("reconcile", "triage")
  g.add_edge("triage", "export")
  g.add_edge("export", END)
  return g.compile()
```

Defines the pipeline

Ingest → Normalize → Reconcile → Triage → Export

Compiled Graph Image



CLI Entry Point

```
def main():
    # Hardcoded defaults
    erp_path = "/home/sudamasharma/aai/app/data/erp_data.xlsx"
    bank_path = "/home/sudamasharma/aai/app/data/bank_statement.pdf"
    out_path = "/home/sudamasharma/aai/app/data/reconciliation_report_by_agent.xlsx"
    tol = 0.05

    app = build_graph()
    initial = RecoState(user_query="Run reconciliation", tol=tol)
```

```
    config = {"configurable": {"erp_path": erp_path, "bank_path": bank_path, "out_path": out_path}}
    final_state = app.invoke(initial, config=config)

    # Print summary & notes
    print("=== SUMMARY ===")
    print(final_state["artifacts"].get("summary"))
    print("\n=== LLM TRIAGE NOTES (optional) ===")
    print(final_state["artifacts"].get("llm_notes", "(skipped)"))
    print(f"\nReport saved to: {out_path}")
```

and it will:

1. Load ERP Excel + Bank PDF
2. Normalize & reconcile
3. Run AI triage (optional)
4. Export Excel report
5. Print results to terminal

Conclusion

The Reconciliation Agent demonstrates a practical integration of **LangChain** and **LangGraph** for orchestrating an AI-driven financial workflow. By embedding **RAG + Agentic AI** into traditional reconciliation, it:

- Normalizes ERP (Excel) and Bank (PDF) data into structured formats.
- Reconciles records with configurable tolerance levels.
- Detects and classifies anomalies with both rule-based logic and LLM reasoning.
- Produces a **multi-sheet Excel output** with matches, mismatches, and AI-driven triage notes.
- Encapsulates steps (ingest → normalize → reconcile → triage → export) into modular LangGraph nodes for **scalability and reusability**.

Why its helpful to the Finance teams to gain the insights through the automation:

The Reconciliation Agent transforms traditional financial reconciliation into a **smart, automated, and audit-ready process**. By combining advanced data processing with **AI-powered reasoning**, it reduces manual work, speeds up month-end close, and improves accuracy.

- **Efficiency** – Faster reconciliation of ERP and bank transactions.
- **Accuracy** – Automatic detection of mismatches, duplicates, and rounding issues.
- **Transparency** – Clear, structured reports ready for auditors and compliance teams.
- **Intelligence** – AI-driven insights into discrepancies, reducing back-and-forth investigations.

1. Situation

Finance teams traditionally face the time-consuming and error-prone challenge of reconciling ERP (Excel) records with Bank statements (PDF). Manual processes often lead to missed discrepancies, delayed month-end closing, and compliance risks. With increasing transaction volumes and tighter audit requirements, organizations need an automated, intelligent, and audit-ready solution.

---

2. Task

The goal was to design and implement an **AI-driven reconciliation system** that:

- Automates the ingestion and normalization of ERP and bank statement data.
- Accurately matches transactions, including handling fuzzy and ambiguous cases.
- Detects and classifies discrepancies (e.g., missing entries, duplicates, rounding variances).

- Provides actionable insights using LLM reasoning (e.g., timing differences, fees).
- Produces structured, multi-sheet reconciliation reports ready for finance and audit teams.

---

3. Action

To achieve this, the solution was built using **LangChain** and **LangGraph** as orchestration frameworks, integrated with RAG (Retrieval-Augmented Generation) and Agentic AI workflows.

Key steps included:

1. **Data Ingestion**
    - Loaded ERP data (Excel) and Bank statements (PDF) using `pandas` and `camelot`.
2. **Normalization**
    - Standardized transaction formats into a unified schema (Date, Description, Invoice, Amount, RefID).
3. **Reconciliation Logic**
    - Applied grouping by invoice, duplicate detection, and merging logic with configurable tolerance (`DEFAULT_TOL=0.05`) to classify matches, mismatches, and rounding differences.
4. **AI Triage (LLM Node)**
    - Used **ChatOpenAI** to analyze unresolved discrepancies and provide contextual reasoning (e.g., FX differences, timing issues, duplicate payments).
5. **Orchestration via LangGraph**
    - Created modular pipeline nodes: **Ingest → Normalize → Reconcile → Triage → Export**.

    - Compiled into a state-driven graph ensuring scalability and reusability.

6. **Report Generation**
    - Exported findings into a multi-sheet Excel report with categories like: Summary, Matched, Amount Mismatch, Missing in Bank/ERP, Duplicates, Adjustments, and AI Notes.

---

4. Result

- **Efficiency**: Reconciliation was automated, reducing manual workload and speeding up financial close processes.
- **Accuracy**: Discrepancies (e.g., missing transactions, duplicates, small variances) were systematically detected.
- **Transparency**: Generated structured, audit-ready reports with clear categorizations.
- **Intelligence**: AI-driven triage added contextual insights, reducing back-and-forth investigations for finance teams.
- **Scalability**: Modular design enables future expansion (e.g., multi-bank integration, continuous learning for anomaly classification).