# Opulence Mainframe Deep Research Agent Architecture

## 1. Simple System Overview

The Opulence system takes legacy mainframe code from a **private wealth bank's security transaction processing system** and makes it understandable using modern AI technology:

- **Input**: COBOL programs, JCL job scripts, PROC procedures, DB2 database definitions, and sample transaction data files from the bank's security trading platform
- **Processing**: Parses and loads them into structured format using a code parser and data loader
- **AI Analysis**: Uses a GPU-hosted CodeLLaMA model (exposed via HTTP API) to analyze and summarize complex business logic
- **Orchestration**: A Coordinator Agent manages the workflow across various specialized research agents
- **Output**: Provides lineage maps showing how customer data flows, business logic summaries explaining trading rules, comprehensive documentation, and an interactive chat interface for asking questions

**Example Scenario**: Understanding how a customer's security purchase order flows through 50+ COBOL programs, what validation rules apply, and how it updates the portfolio database.

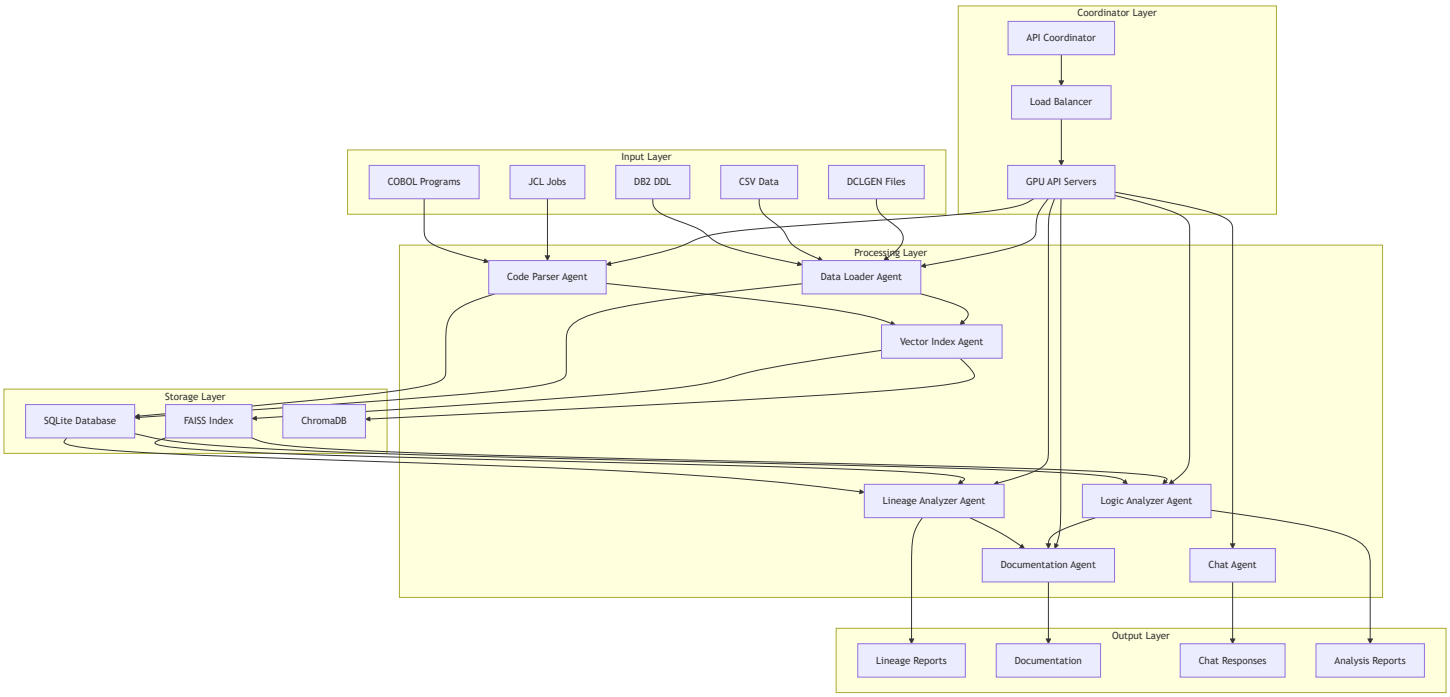
## 2. Core Components

| Component | Function | Value |
|---|---|---|
| **Code Parser** | Converts COBOL/JCL into structured AST | Enables structured understanding of 40-year-old logic |
| **Data Loader** | Loads DB2 tables and sample transaction files | Adds real-world context from actual customer trades |
| **Vector Index Agent** | Embeds and indexes all elements in FAISS | Powers fast semantic search: "find all margin calculation logic" |

| Component | Function | Value |
|---|---|---|
| **Lineage Agent** | Tracks fields across jobs and programs | Critical for compliance: trace customer ID through entire system |
| **Logic Analyzer Agent** | Extracts business logic and conditional rules | Automates discovery of trading rules and validation logic |
| **Documentation Agent** | Summarizes components and logic | Generates readable docs explaining arcane settlement processes |
| **Chat Agent** | Interfaces with user to answer questions | "How does stop-loss order processing work?" gets instant answers |
| **Coordinator Agent** | Orchestrates flow and agent sequencing | Ensures systematic analysis of interconnected trading systems |
| **GPU LLM API** | CodeLLaMA exposed via API for summarization | Core intelligence for understanding legacy financial code |

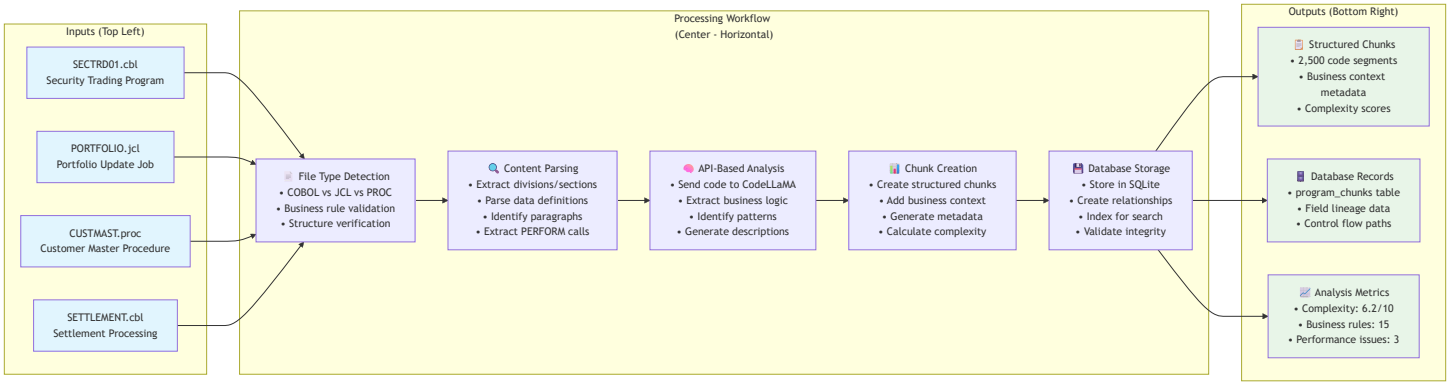
# 3. System Flow and Individual Agent Workflows

## Overall System Architecture Flow

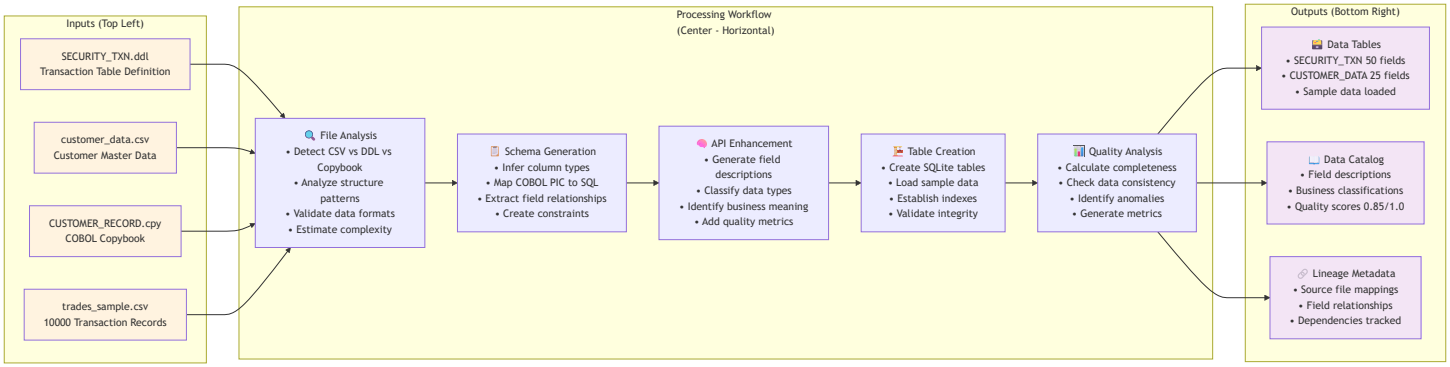


# 4. Individual Agent Workflows

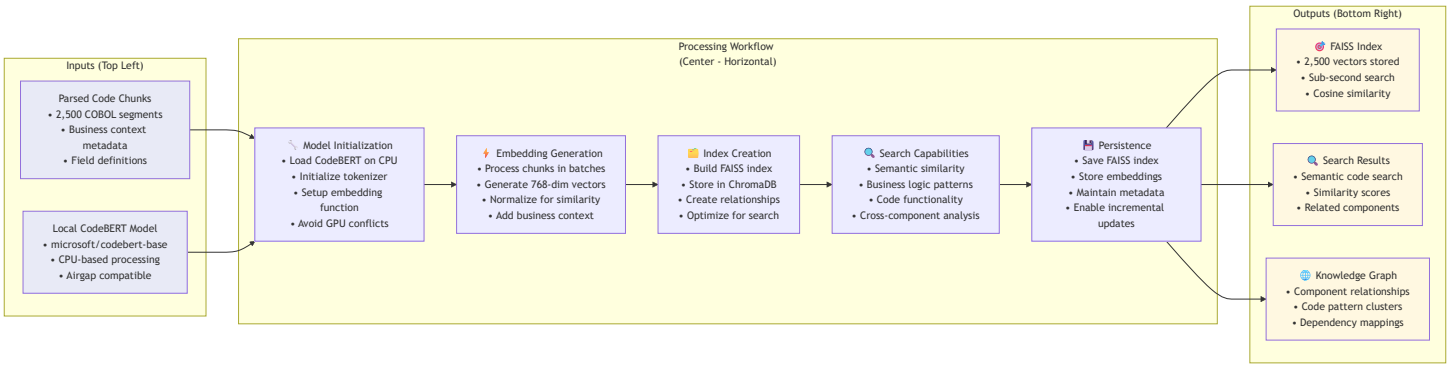
## 4.1 Code Parser Agent Flow



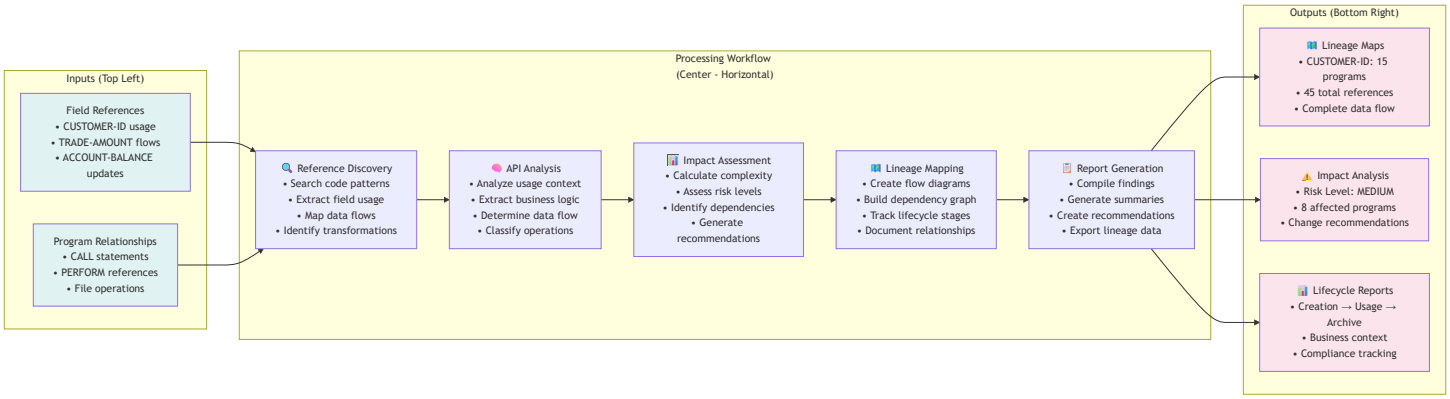

## 4.2 Data Loader Agent Flow



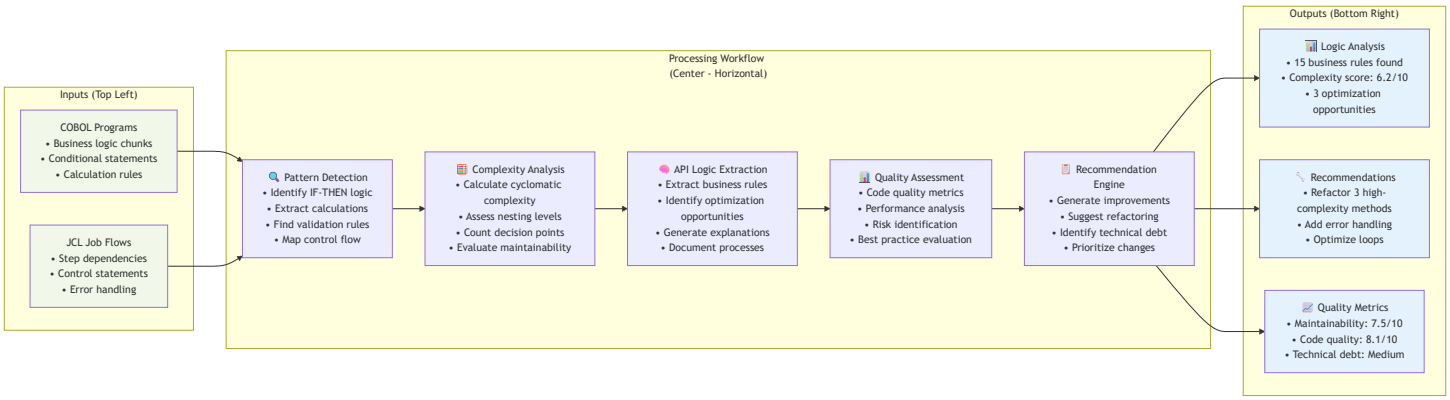

## 4.3 Vector Index Agent Flow



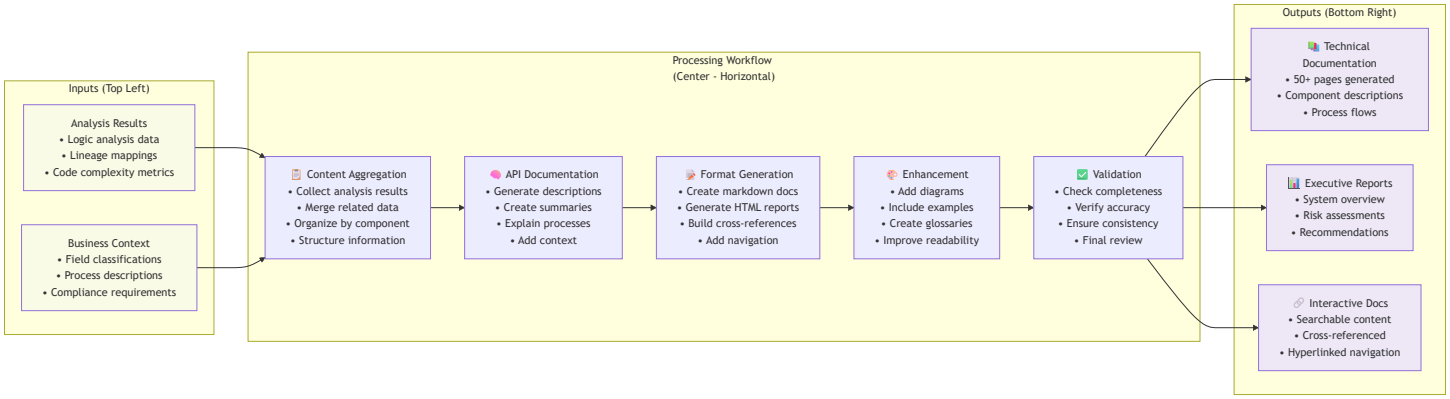
# 4.4 Lineage Analyzer Agent Flow



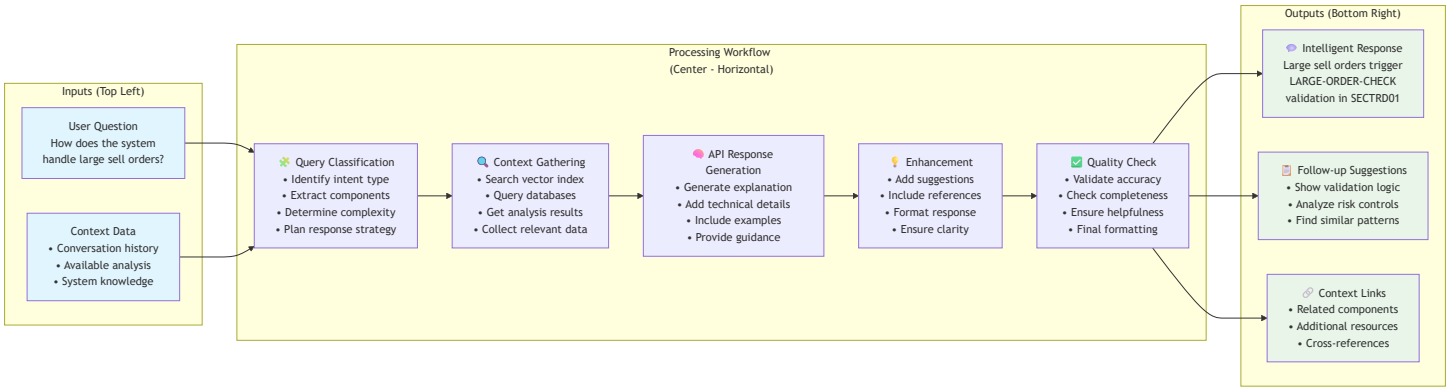

# 4.5 Logic Analyzer Agent Flow



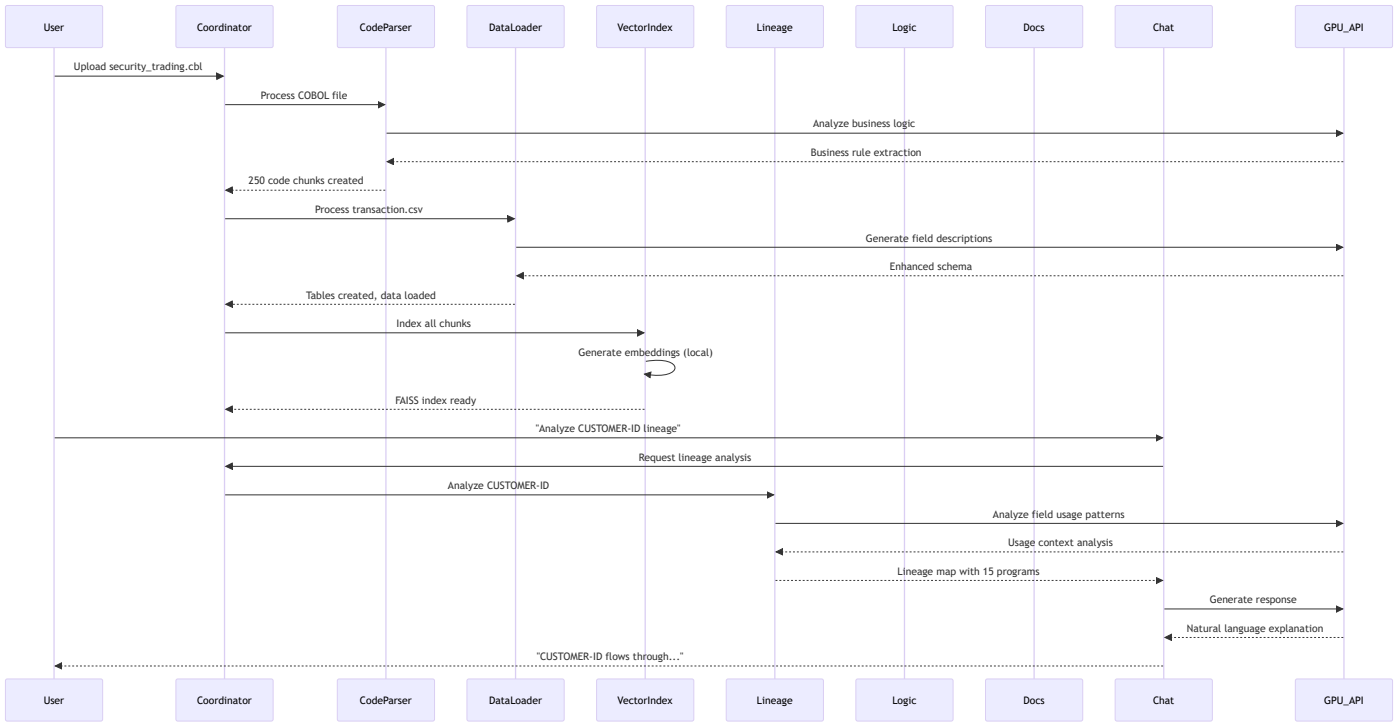

# 4.6 Documentation Agent Flow



# 4.7 Chat Agent Flow




# 5. Agent Coordination Flow




# 6. Output Artifacts

The Opulence system produces these deliverables for the bank's security trading system:

### ✅ Field-level data lineage reports

- "CUSTOMER-ID flows from CUSTMAST → SECTRD01 → PORTFOLIO-UPDATE → TRADE-HISTORY"
- Compliance-ready audit trails

✅ **Extracted business logic summaries**

- "Stop-loss orders: IF CURRENT-PRICE < (STOP-PRICE * 0.95) THEN EXECUTE-SELL"
- Trading rule documentation in plain English

✅ **Annotated markdown documentation of code modules**

- Complete explanation of settlement processing
- Cross-references between related programs

✅ **Interactive chat interface for querying understanding**

- "What happens when a trade fails settlement?"
- "Show me all programs that update customer portfolios"

# 7. Sample Data Context: Private Wealth Bank Security Transactions

## Input Files for Analysis:

**COBOL Programs:**

- `SECTRD01.cbl` - Main security trading program (2,500 lines)
- `VALIDATE.cbl` - Order validation logic (800 lines)
- `SETTLE.cbl` - Settlement processing (1,200 lines)
- `PORTFOLIO.cbl` - Portfolio update logic (900 lines)

**JCL Jobs:**

- `DAILYTRD.jcl` - Daily trade processing batch job
- `SETTLEMENT.jcl` - End-of-day settlement job
- `RECON.jcl` - Trade reconciliation job

**DB2 Tables:**

```
-- SECURITY_TRANSACTION table
CREATE TABLE SECURITY_TXN (
    CUST_ID        CHAR(10),
    TRADE_ID       CHAR(15),
    SECURITY_CODE  CHAR(8),
    TRADE_TYPE     CHAR(4),    -- BUY/SELL
    QUANTITY       DECIMAL(15,2),
    PRICE          DECIMAL(15,4),
    TRADE_DATE     DATE,
    SETTLE_DATE    DATE,
    STATUS         CHAR(3)     -- PEN/SET/FAI
);
```

**Sample Transaction Data:**

```
CUST_ID,TRADE_ID,SECURITY_CODE,TRADE_TYPE,QUANTITY,PRICE,TRADE_DATE,STATUS
PWB0001234,TRD20241201001,AAPL,BUY,100,150.25,2024-12-01,PEN
PWB0001234,TRD20241201002,TSLA,SELL,50,245.80,2024-12-01,SET
PWB0001567,TRD20241201003,MSFT,BUY,200,380.15,2024-12-01,FAI
```

# 8. Individual Agent Explanations

## Vector Index Agent

**Purpose**: Creates searchable embeddings of all code segments and business logic.

**Bank Example**: When analyzing the security trading system, this agent:

- Embeds all COBOL paragraphs dealing with order validation
- Creates vectors for trading rule conditions
- Enables semantic search like "find all margin calculation logic"

**API Integration**: Makes HTTP calls to CodeLLaMA to generate embeddings and understand code semantics.

## Lineage Agent

**Purpose**: Tracks how data fields flow through the entire system.

**Bank Example**: For a customer security purchase:

1. **CUSTOMER-ID** enters via online trading platform
2. Flows through `VALIDATE.cbl` for KYC checks
3. Processed in `SECTRD01.cbl` for order execution
4. Updates `PORTFOLIO.cbl` for position management
5. Records in `TRADE-HISTORY` table for audit

**Critical for Compliance**: Regulators require complete audit trails showing how customer data is processed.

## Logic Analyzer Agent

**Purpose**: Extracts and explains complex business rules embedded in COBOL logic.

**Bank Example**: Discovers trading rules like:

```
IF TRADE-AMOUNT > DAILY-LIMIT
   AND CUSTOMER-TIER NOT = 'PLATINUM'
   THEN MOVE 'HOLD' TO TRADE-STATUS
   PERFORM MANUAL-APPROVAL-PROCESS
```

Translates to: "Trades over daily limit require manual approval unless customer is Platinum tier."

## Documentation Agent

**Purpose**: Creates human-readable documentation explaining system functionality.

**Bank Example**: Generates documentation like:

- "Settlement Process Overview: How T+2 settlement works"
- "Stop-Loss Order Processing: Automated selling when price thresholds are breached"
- "Customer Portfolio Updates: Real-time vs. batch processing logic"

## Chat Agent

**Purpose**: Provides conversational interface for querying system knowledge.

**Bank Example Queries**:

- "How does the system handle partial fills on large orders?"
- "What validation checks are performed before executing a trade?"

- "Show me the settlement process for international securities"

**Response Example**: "When a large order cannot be filled completely, the PARTIAL-FILL-HANDLER in SECTRD01 splits it into smaller chunks and processes them separately, updating the customer's available cash after each partial execution..."

# 9. Coordination Flow: Processing a Security Transaction

## Real-World Scenario: Customer Places $500K Apple Stock Purchase

1. **File Processing Phase**:
   - Code Parser analyzes `SECTRD01.cbl` and extracts order processing logic
   - Data Loader imports recent Apple trading data and customer portfolio info
   - System identifies all programs involved in large order processing
2. **Analysis Phase**:
   - **Vector Index Agent**: Finds all code segments related to large order handling
   - **Lineage Agent**: Maps how customer cash balance flows through the system
   - **Logic Analyzer**: Extracts validation rules for large orders (credit checks, position limits)
   - **Documentation Agent**: Summarizes the complete order-to-settlement workflow
3. **Query Phase**:
   - Risk manager asks: "What approvals are needed for this trade size?"
   - Chat Agent searches indexed knowledge and responds: "Orders over $250K require senior trader approval per LARGE-ORDER-CHECK paragraph, plus real-time margin calculation..."
4. **Compliance Phase**:
   - Lineage reports show complete audit trail
   - Logic summaries document all decision points
   - Documentation provides regulatory-compliant process descriptions

This architecture transforms decades-old, undocumented mainframe code into an accessible, searchable knowledge base that supports both operational teams and regulatory compliance requirements.

# 10. Technical Implementation Notes

## API-Based Architecture

The Opulence system uses HTTP APIs to communicate with GPU-hosted CodeLLaMA models, enabling:

- **Scalability**: Multiple model servers can handle concurrent analysis requests
- **Load Balancing**: Requests are distributed across available GPU resources
- **Fault Tolerance**: Circuit breakers and retry logic ensure robust operation
- **Resource Efficiency**: No need for local GPU allocation per agent

## Database Design

SQLite database stores:

- **program_chunks**: Parsed code segments with metadata
- **field_lineage**: Data flow tracking for compliance
- **vector_embeddings**: FAISS index references for semantic search
- **processing_stats**: Performance monitoring and audit trails

This architecture enables financial institutions to understand and maintain critical legacy systems while meeting modern regulatory and operational requirements.