Opulence Mainframe Deep Research Agent Architecture

1. Simple System Overview

The Opulence system has been enhanced to take legacy mainframe code, data and file structure and makes it understandable program logic in a structured manner, data flow within the subsystem using legacy mainframe code and data storage from vsam files, to determine if the usage processing for files and programs, field usage and duplication of fields and files and determine obsolete or duplicated data structures.

- **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.

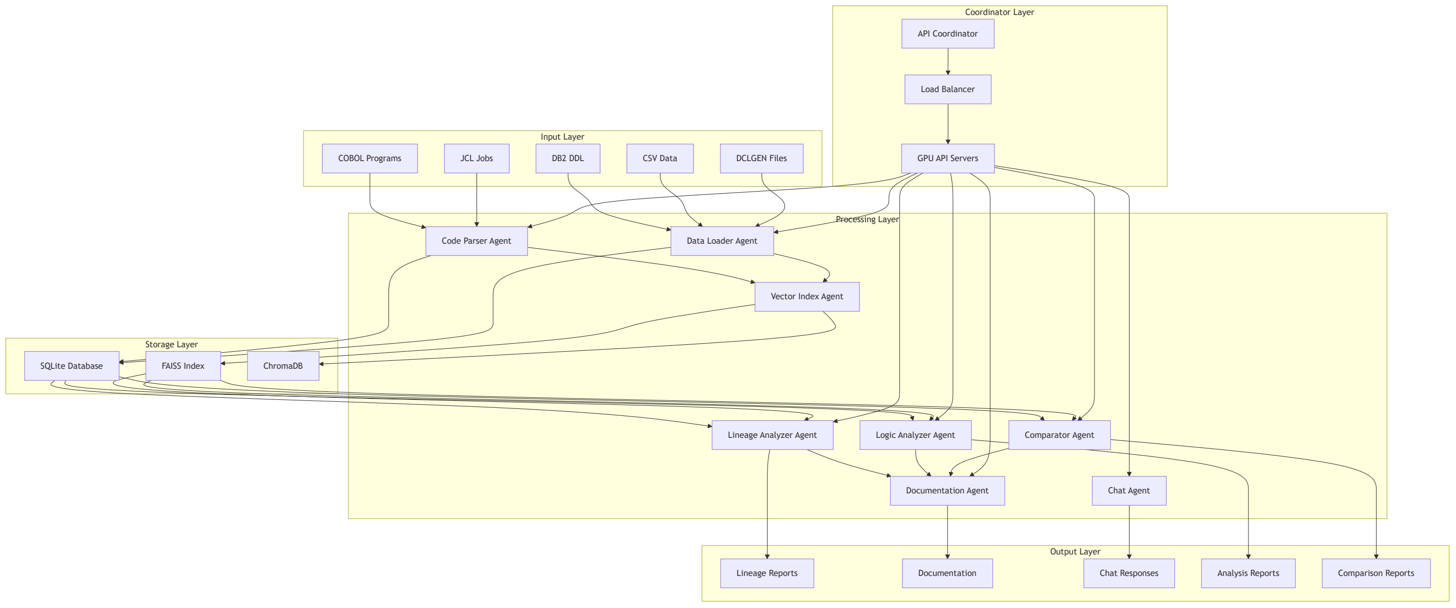
2. Core Components

Component	Function	Value
Code Parser	Converts COBOL/JCL into structured AST , store in Sqlite DB	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"
Lineage Agent	Tracks fields across jobs and programs	Critical for compliance: trace customer ID through entire system

Component	Function	Value
Logic Analyzer Agent	Extracts business logic and conditional rules	Automates discovery of trading rules and validation logic
Comparator Agent	Compares similar files and identifies patterns	Finds duplicate logic, unused fields, and optimization opportunities
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

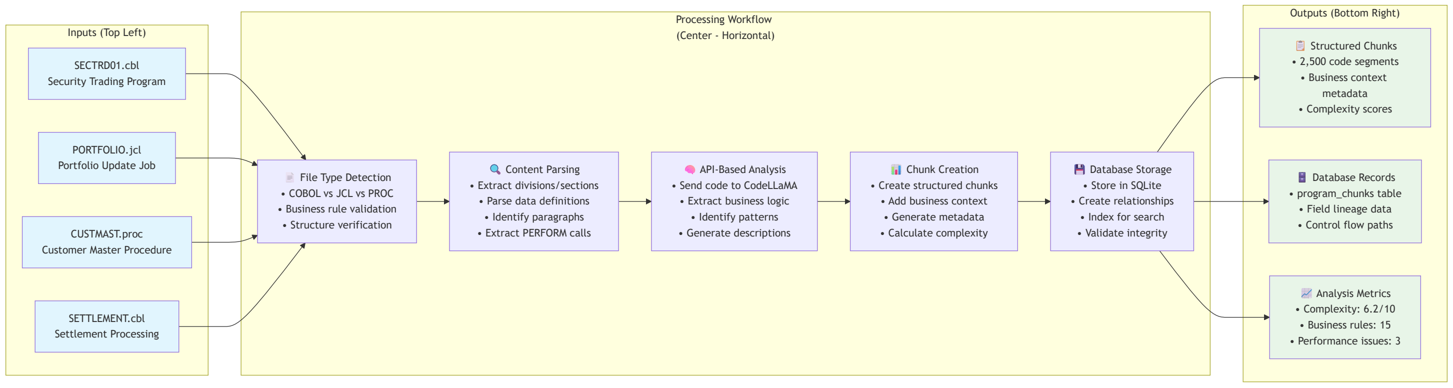
3. System Flow and Individual Agent Workflows

Overall System Architecture Flow



4. Individual Agent Workflows

4.1 Code Parser Agent Flow


Inputs (Top Left)
SECTRD01.cbl Security Trading Program
PORTFOLIO.jcl Portfolio Update Job
CUSTMAST.proc Customer Master Procedure
SETTLEMENT.cbl Settlement Processing
Processing Workflow (Center - Horizontal)
File Type Detection • COBOL vs JCL vs PROC • Business rule validation • Structure verification
Content Parsing • Extract divisions/sections • Parse data definitions • Identify paragraphs • Extract PERFORM calls
API-Based Analysis • Send code to CodeLLaMA • Extract business logic • Identify patterns • Generate descriptions
Chunk Creation • Create structured chunks • Add business context • Generate metadata • Calculate complexity
Database Storage • Store in SQLite • Create relationships • Index for search • Validate integrity
Outputs (Bottom Right)
Structured Chunks • 2,500 code segments • Business context metadata • Complexity scores
Database Records • program_chunks table • Field lineage data • Control flow paths
Analysis Metrics • Complexity: 6.2/10 • Business rules: 15 • Performance issues: 3

Sample Output:

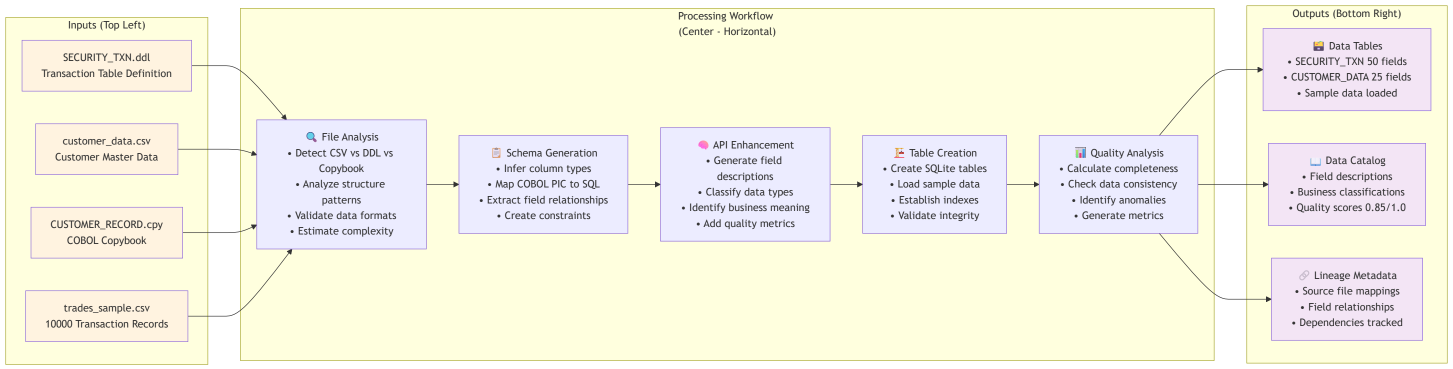
Program Name: SECTRD01.cbl

- Total chunks: 156
- Complexity score: 6.2 out of 10
- Business rules found: 15
- Performance issues: 3

Key sections include:

- VALIDATE-ORDER section (lines 245-387)
  - Complexity: 8.1
  - Business logic: Validates customer orders against credit limits and risk parameters

4.2 Data Loader Agent Flow


Inputs (Top Left)
SECURITY_TXN.ddl Transaction Table Definition
customer_data.csv Customer Master Data
CUSTOMER_RECORD.cpy COBOL Copybook
trades_sample.csv 10000 Transaction Records
Processing Workflow (Center - Horizontal)
File Analysis • Detect CSV vs DDL vs Copybook • Analyze structure patterns • Validate data formats • Estimate complexity
Schema Generation • Infer column types • Map COBOL PIC to SQL • Extract field relationships • Create constraints
API Enhancement • Generate field descriptions • Classify data types • Identify business meaning • Add quality metrics
Table Creation • Create SQLite tables • Load sample data • Establish indexes • Validate integrity
Quality Analysis • Calculate completeness • Check data consistency • Identify anomalies • Generate metrics
Outputs (Bottom Right)
Data Tables • SECURITY_TXN 50 fields • CUSTOMER_DATA 25 fields • Sample data loaded
Data Catalog • Field descriptions • Business classifications • Quality scores 0.85/1.0
Lineage Metadata • Source file mappings • Field relationships • Dependencies tracked

Sample Output:

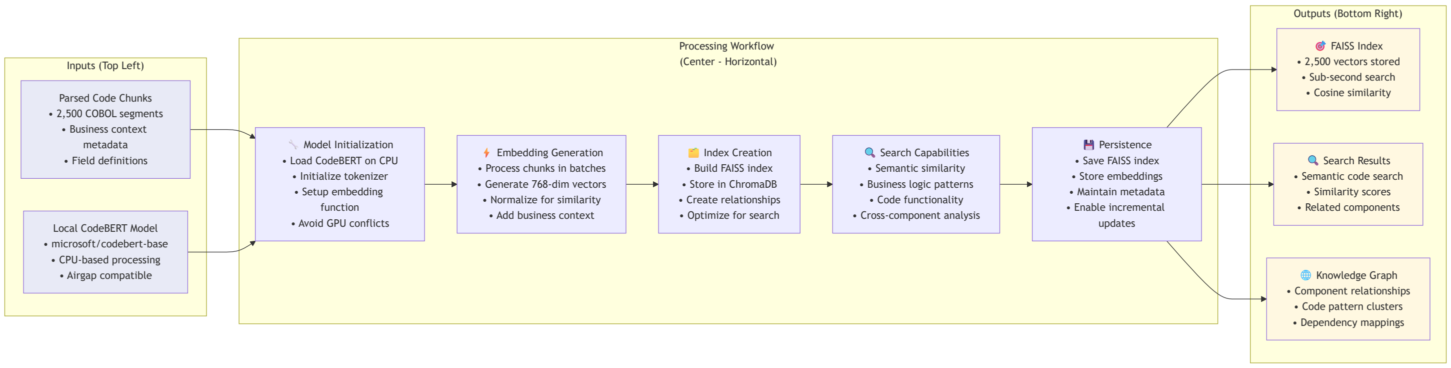
Table Name: SECURITY_TXN

- Total fields: 50
- Data quality score: 0.85 out of 1.0
- Loaded records: 10,000

Field Classifications:

- CUSTOMER_ID: CHAR(10), Primary customer identifier, Business category: Customer Reference, Completeness: 100%
- TRADE_AMOUNT: DECIMAL(15,2), Total trade value in USD, Business category: Financial, Completeness: 98%

4.3 Vector Index Agent Flow


Inputs (Top Left)
Parsed Code Chunks • 2,500 COBOL segments • Business context metadata • Field definitions
Local CodeBERT Model • microsoft/codebert-base • CPU-based processing • Airgap compatible
Processing Workflow (Center - Horizontal)
Model Initialization • Load CodeBERT on CPU • Initialize tokenizer • Setup embedding function • Avoid GPU conflicts
Embedding Generation • Process chunks in batches • Generate 768-dim vectors • Normalize for similarity • Add business context
Index Creation • Build FAISS index • Store in ChromaDB • Create relationships • Optimize for search
Search Capabilities • Semantic similarity • Business logic patterns • Code functionality • Cross-component analysis
Persistence • Save FAISS index • Store embeddings • Maintain metadata • Enable incremental updates
Outputs (Bottom Right)
FAISS Index • 2,500 vectors stored • Sub-second search • Cosine similarity
Search Results • Semantic code search • Similarity scores • Related components
Knowledge Graph • Component relationships • Code pattern clusters • Dependency mappings

Sample Output:

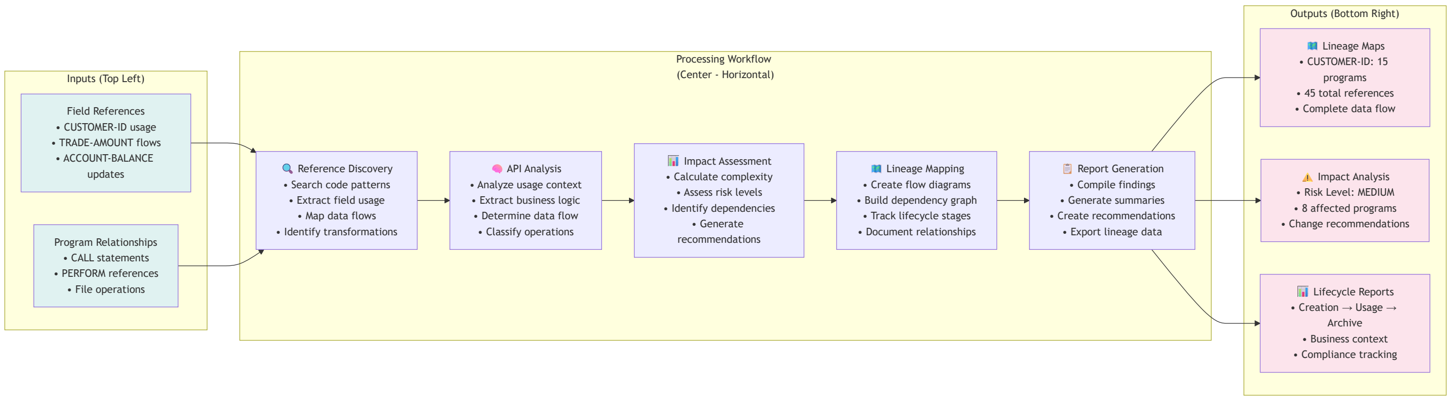
Index Statistics:

- Total vectors: 2,500
- Embedding dimension: 768
- Index size: 45.2 MB
- Search time: 23 milliseconds

Search Example for "customer credit validation":

- Chunk ID: SECTRD01_245_387
- Similarity score: 0.94
- Content: VALIDATE-CUSTOMER-CREDIT section
- Program: SECTRD01.cbl

4.4 Lineage Analyzer Agent Flow


Inputs (Top Left)
Field References • CUSTOMER-ID usage • TRADE-AMOUNT flows • ACCOUNT-BALANCE updates
Program Relationships • CALL statements • PERFORM references • File operations
Processing Workflow (Center - Horizontal)
🔍 Reference Discovery • Search code patterns • Extract field usage • Map data flows • Identify transformations
🧠 API Analysis • Analyze usage context • Extract business logic • Determine data flow • Classify operations
📊 Impact Assessment • Calculate complexity • Assess risk levels • Identify dependencies • Generate recommendations
🗺 Lineage Mapping • Create flow diagrams • Build dependency graph • Track lifecycle stages • Document relationships
📋 Report Generation • Compile findings • Generate summaries • Create recommendations • Export lineage data
Outputs (Bottom Right)
🗺 Lineage Maps • CUSTOMER-ID: 15 programs • 45 total references • Complete data flow
⚠️ Impact Analysis • Risk Level: MEDIUM • 8 affected programs • Change recommendations
📊 Lifecycle Reports • Creation → Usage → Archive • Business context • Compliance tracking

Sample Output:

Field Name: CUSTOMER_ID

- Total references: 45
- Programs affected: 15

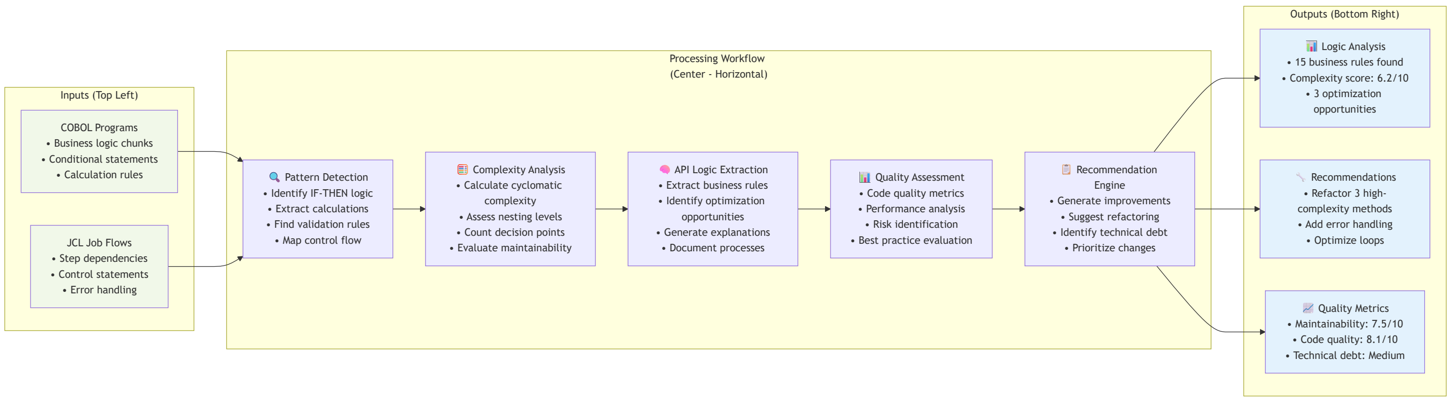
Lineage Flow:

1. CUSTMAST.cbl (line 156): CREATE - Initial customer registration
2. SECTRD01.cbl (line 245): READ - Order validation lookup
3. PORTFOLIO.cbl (line 389): UPDATE - Portfolio balance update

Impact Analysis:

- Risk level: MEDIUM
- Change complexity: 7.2 out of 10
- Affected business processes: 8

4.5 Logic Analyzer Agent Flow


Inputs (Top Left)
COBOL Programs • Business logic chunks • Conditional statements • Calculation rules
JCL Job Flows • Step dependencies • Control statements • Error handling
Processing Workflow (Center - Horizontal)
🔍 Pattern Detection • Identify IF-THEN logic • Extract calculations • Find validation rules • Map control flow
📋 Complexity Analysis • Calculate cyclomatic complexity • Assess nesting levels • Count decision points • Evaluate maintainability
🧠 API Logic Extraction • Extract business rules • Identify optimization opportunities • Generate explanations • Document processes
📊 Quality Assessment • Code quality metrics • Performance analysis • Risk identification • Best practice evaluation
📋 Recommendation Engine • Generate improvements • Suggest refactoring • Identify technical debt • Prioritize changes
Outputs (Bottom Right)
📊 Logic Analysis • 15 business rules found • Complexity score: 6.2/10 • 3 optimization opportunities
🔧 Recommendations • Refactor 3 high-complexity methods • Add error handling • Optimize loops
📈 Quality Metrics • Maintainability: 7.5/10 • Code quality: 8.1/10 • Technical debt: Medium

Sample Output:

Program Name: SECTRD01.cbl

- Business rules extracted: 15
- Complexity metrics:
  - Cyclomatic complexity: 6.2
  - Nesting levels: 4
  - Decision points: 23
  - Maintainability score: 7.5 out of 10

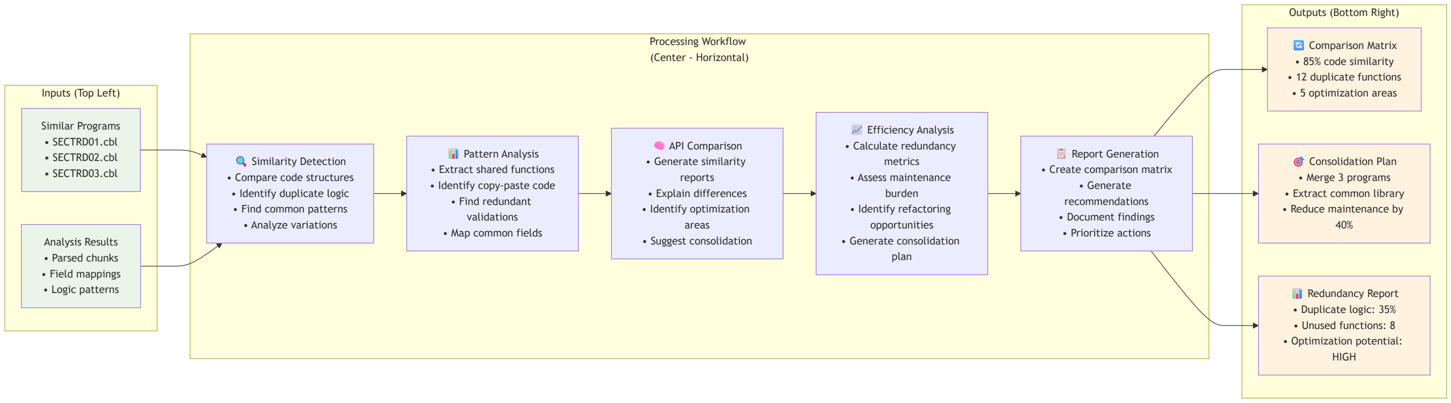
Extracted Rules Example:

- Rule ID: LARGE_ORDER_CHECK
- Condition: IF TRADE-AMOUNT > 250000
- Action: PERFORM MANUAL-APPROVAL-PROCESS
- Business context: Orders over $250K require manual approval

Recommendations:

- Type: REFACTOR
- Priority: HIGH
- Description: Break down VALIDATE-ORDER section - too complex

4.6 Comparator Agent Flow


Inputs (Top Left)
Similar Programs • SECTRD01.cbl • SECTRD02.cbl • SECTRD03.cbl
Analysis Results • Parsed chunks • Field mappings • Logic patterns
Processing Workflow (Center - Horizontal)
Similarity Detection • Compare code structures • Identify duplicate logic • Find common patterns • Analyze variations
Pattern Analysis • Extract shared functions • Identify copy-paste code • Find redundant validations • Map common fields
API Comparison • Generate similarity reports • Explain differences • Identify optimization areas • Suggest consolidation
Efficiency Analysis • Calculate redundancy metrics • Assess maintenance burden • Identify refactoring opportunities • Generate consolidation plan
Report Generation • Create comparison matrix • Generate recommendations • Document findings • Prioritize actions
Outputs (Bottom Right)
Comparison Matrix • 85% code similarity • 12 duplicate functions • 5 optimization areas
Consolidation Plan • Merge 3 programs • Extract common library • Reduce maintenance by 40%
Redundancy Report • Duplicate logic: 35% • Unused functions: 8 • Optimization potential: HIGH

Sample Output:

Comparison Summary:

- Files compared: SECTRD01.cbl, SECTRD02.cbl, SECTRD03.cbl
- Similarity score: 0.85 (85%)
- Duplicate functions: 12
- Redundant code percentage: 35%

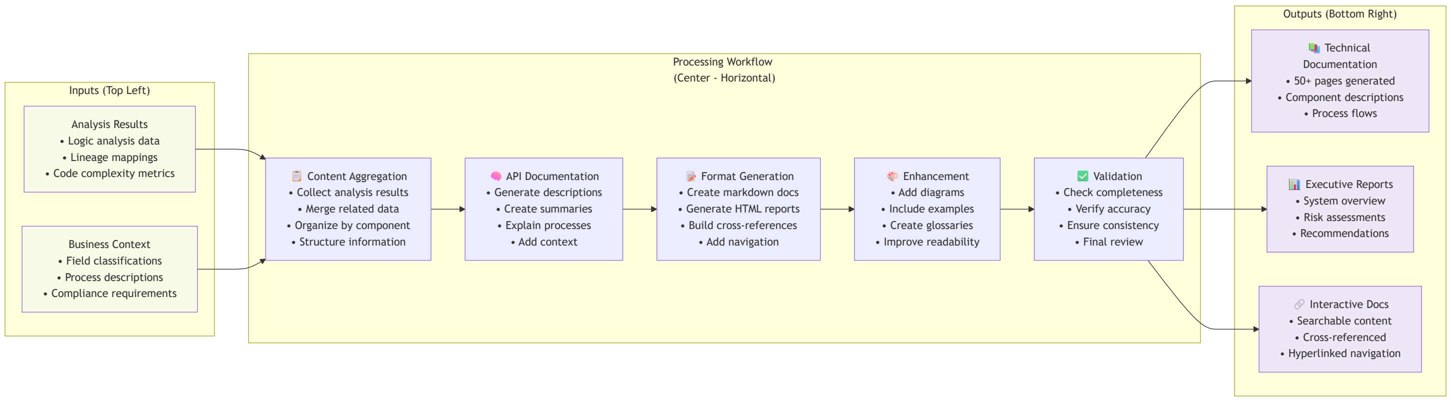
Duplicate Patterns:

- Pattern: VALIDATE-CUSTOMER-CREDIT
- Occurrences: 3 files
- Consolidation opportunity: HIGH

Optimization Recommendations:

- Type: EXTRACT_COMMON_LIBRARY
- Description: Create shared validation library
- Estimated effort: 3 weeks
- Maintenance reduction: 40%

4.7 Documentation Agent Flow


Inputs (Top Left)
Analysis Results • Logic analysis data • Lineage mappings • Code complexity metrics
Business Context • Field classifications • Process descriptions • Compliance requirements
Processing Workflow (Center - Horizontal)
📋 Content Aggregation • Collect analysis results • Merge related data • Organize by component • Structure information
🧠 API Documentation • Generate descriptions • Create summaries • Explain processes • Add context
📝 Format Generation • Create markdown docs • Generate HTML reports • Build cross-references • Add navigation
🎨 Enhancement • Add diagrams • Include examples • Create glossaries • Improve readability
✅ Validation • Check completeness • Verify accuracy • Ensure consistency • Final review
Outputs (Bottom Right)
📚 Technical Documentation • 50+ pages generated • Component descriptions • Process flows
📊 Executive Reports • System overview • Risk assessments • Recommendations
🔗 Interactive Docs • Searchable content • Cross-referenced • Hyperlinked navigation

Sample Output:

Documentation Summary:

- Total pages: 52
- Sections generated: 8
- Cross-references: 134
- Diagrams created: 15

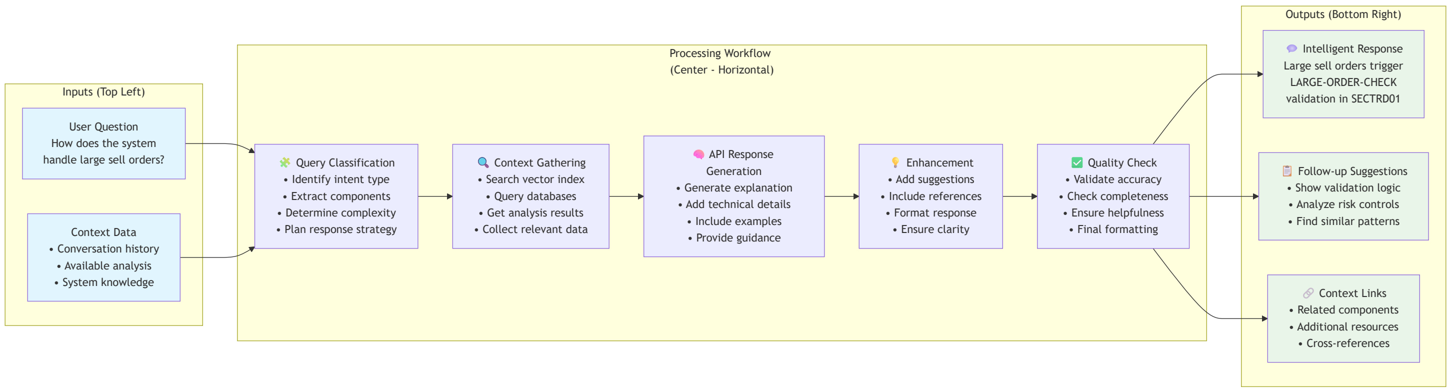
Document Sections:

1. Security Trading System Overview (8 pages)
   - Content type: Executive Summary
   - Key topics: System Architecture, Business Processes, Risk Assessment
2. SECTRD01 Program Analysis (12 pages)
   - Content type: Technical Deep Dive
   - Key topics: Logic Flow, Business Rules, Performance Analysis

Quality Metrics:

- Completeness: 92%
- Accuracy: 88%
- Readability score: 8.5 out of 10

4.8 Chat Agent Flow



Sample Output:

Query: "How does the system handle large sell orders?"

Response:

- Main answer: Large sell orders trigger the LARGE-ORDER-CHECK validation in SECTRD01.cbl. Orders over $250K require manual approval and additional risk assessment.

Technical Details:

- Validation occurs in paragraph VALIDATE-LARGE-ORDER (lines 245-387)
- Risk assessment includes margin calculation and position limits
- Manual approval workflow triggers EMAIL-NOTIFICATION process

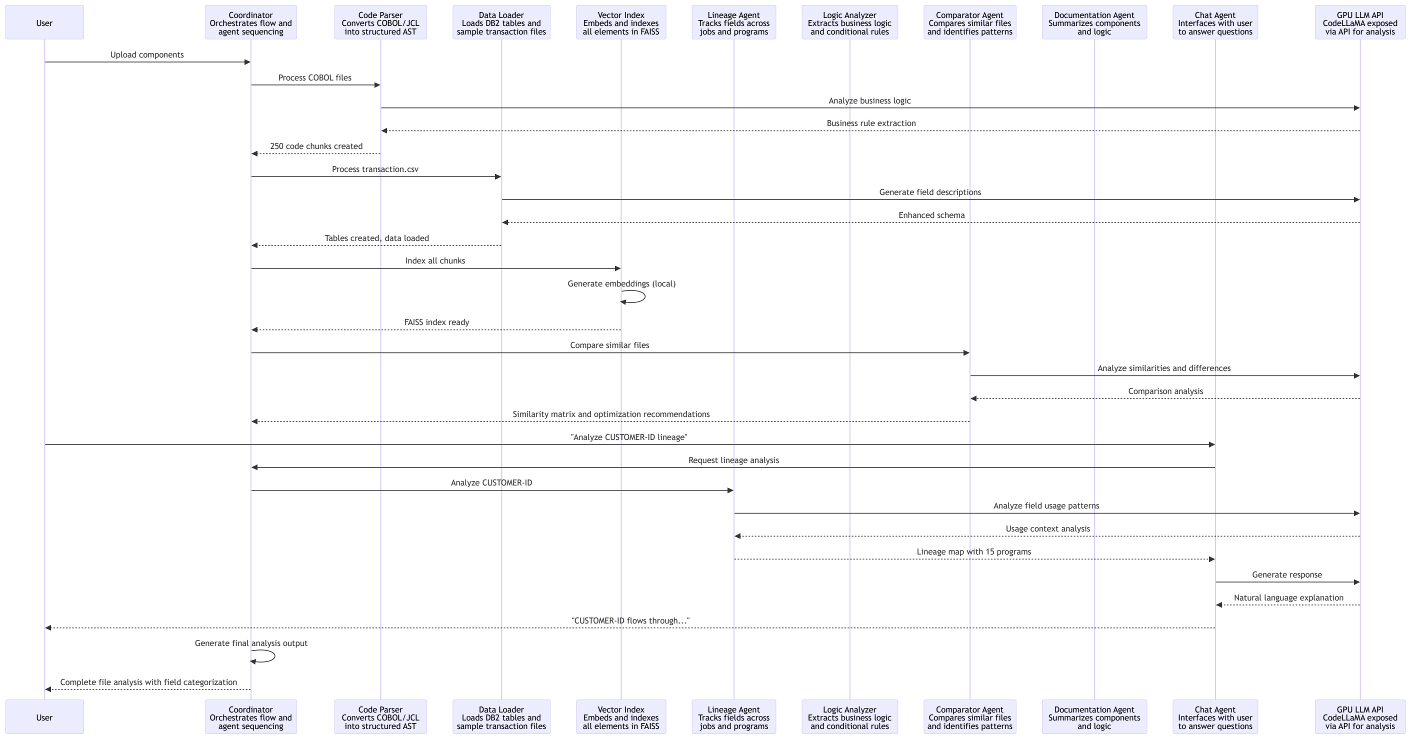
Related Components:

- SECTRD01.cbl - Main validation logic
- RISKMGMT.cbl - Risk assessment calculations
- APPROVAL.cbl - Manual approval workflow

Follow-up Suggestions:

- Show me the exact validation logic
- What are the risk thresholds?
- How long does manual approval take?

5. Agent Coordination Flow


User
Coordinator Orchestrates flow and agent sequencing
Code Parser Converts COBOL/JCL into structured AST
Data Loader Loads DB2 tables and sample transaction files
Vector Index Embeds and indexes all elements in FAISS
Lineage Agent Tracks fields across jobs and programs
Logic Analyzer Extracts business logic and conditional rules
Comparator Agent Compares similar files and identifies patterns
Documentation Agent Summarizes components and logic
Chat Agent Interfaces with user to answer questions
GPU LLM API CodeLLaMA exposed via API for analysis
Upload components
Process COBOL files
Analyze business logic
Business rule extraction
250 code chunks created
Process transaction.csv
Generate field descriptions
Enhanced schema
Tables created, data loaded
Index all chunks
Generate embeddings (local)
FAISS index ready
Compare similar files
Analyze similarities and differences
Comparison analysis
Similarity matrix and optimization recommendations
"Analyze CUSTOMER-ID lineage"
Request lineage analysis
Analyze CUSTOMER-ID
Analyze field usage patterns
Usage context analysis
Lineage map with 15 programs
Generate response
Natural language explanation
"CUSTOMER-ID flows through..."
Generate final analysis output
Complete file analysis with field categorization

6. Final Analysis Output Structure

The Coordinator Agent produces a comprehensive file analysis report with the following structure:

6.1 Field Classification Analysis Summary

Overall Statistics:

- Files processed: 15
- Total fields analyzed: 347
- Analysis completed: December 1, 2024

Field Categories Overview:

Fields from Input (125 total)

These are fields that originate from external sources and user inputs.

Key Statistics:

- Simple fields: 78
- Complex fields: 47
- Average complexity score: 4.2 out of 10
- Data quality rate: 94% complete

Category Breakdown:

Calculated Fields (67 fields)
Fields that result from mathematical operations and business logic.

Examples:

- TOTAL_COMMISSION (SECTRD01.cbl, line 456)
  - Logic: TRADE_AMOUNT * COMMISSION_RATE / 100
  - Complexity: 4.5, Performance impact: LOW
  - Business rule: Commission calculated as percentage of trade amount
- NET_SETTLEMENT (SETTLE.cbl, line 234)
  - Logic: TRADE_AMOUNT - TOTAL_COMMISSION - FEES - TAXES + REBATES
  - Complexity: 6.8, Performance impact: MEDIUM
  - Business rule: Final settlement amount after all deductions
- PORTFOLIO_WEIGHTED_RISK (RISKMGMT.cbl, line 567)
  - Logic: Complex risk algorithm with 15+ variables
  - Complexity: 9.2, Performance impact: HIGH
  - Business rule: Sophisticated portfolio risk using Monte Carlo simulation

Status Update Fields (45 fields)
Fields that track state changes and workflow progression.

Examples:

- ORDER_STATUS (VALIDATE.cbl, line 189)
  - Logic: IF VALIDATION_PASSED AND CREDIT_CHECK_OK THEN 'APPROVED' ELSE 'REJECTED'
  - Complexity: 5.2, States: PENDING, VALIDATING, APPROVED, REJECTED, ON_HOLD
  - Business rule: Status updated based on comprehensive validation results
- SETTLEMENT_STATUS (SETTLE.cbl, line 345)
  - Logic: Complex workflow with T+2 settlement rules and exception handling
  - Complexity: 7.4, States: PENDING_SETTLEMENT, SETTLING, SETTLED, FAILED, REVERSED
  - Business rule: Multi-stage settlement process with regulatory compliance

Derived Fields (44 fields)
Fields created through aggregation, transformation, and enrichment processes.

Examples:

- RISK_SCORE (RISKMGMT.cbl, line 123)
    - Logic: CUSTOMER_TIER_WEIGHT * 0.3 + TRADE_SIZE_FACTOR * 0.4 + VOLATILITY_INDEX * 0.3
    - Complexity: 6.5, Algorithm: Weighted scoring
    - Business rule: Composite risk assessment for trade approval
- CUSTOMER_LIFETIME_VALUE (ANALYTICS.cbl, line 678)
    - Logic: Advanced CLV calculation using historical data and predictive modeling
    - Complexity: 8.9, Algorithm: Predictive analytics with ML components
    - Business rule: Predictive customer lifetime value using 5-year analysis

Fields Unused and Static (66 total)

Fields that are no longer actively used or contain static reference data.

Key Statistics:

- Simple static fields: 42
- Complex obsolete fields: 24
- Average obsolescence age: 4.2 years
- Storage overhead: 15.6 MB

Category Breakdown:

Obsolete Fields (24 fields)
Fields that are no longer used due to system changes or business process updates.

Examples:

- OLD_ACCOUNT_TYPE (LEGACY.cpy)
    - Last used: March 15, 2018
    - Complexity: 3.2, Removal effort: 3 days
    - Reason: Replaced by NEW_ACCOUNT_CLASSIFICATION system in 2019
    - Risk level: LOW - Safe to remove
- MANUAL_OVERRIDE_CODE (SECTRD01.cbl)
    - Last used: August 22, 2019
    - Complexity: 6.7, Removal effort: 2 weeks
    - Reason: Automated processing eliminated manual overrides in 2020

    - Risk level: MEDIUM - Review required due to audit trail dependencies
- LEGACY_SETTLEMENT_METHOD (OLDSETTLE.cbl)
    - Last used: November 30, 2017
    - Complexity: 8.4, Removal effort: 3 weeks
    - Reason: Pre-T+2 settlement method, no longer regulatory compliant
    - Risk level: HIGH - Requires deep analysis due to regulatory history

Static Reference Data (28 fields)
Fields containing constant values that could be externalized to configuration.

Examples:

- COMPANY_TAX_ID: Fixed value "12-3456789"
    - Current storage: Hardcoded
    - Recommendation: Move to configuration table
    - Benefit: Centralized management
- SETTLEMENT_DAYS: Fixed value "2"
    - Current storage: Hardcoded
    - Recommendation: Externalize to business rules engine
    - Benefit: Dynamic configuration for T+2 standard

Unused Declared Fields (14 fields)
Fields declared in working storage but never referenced in program logic.

Examples:

- BACKUP_PROCESSING_FLAG (SECTRD01.cbl, line 78)
    - Status: Declared but never used
    - Removal effort: 30 minutes
    - Risk level: VERY LOW - Safe to remove
- FUTURE_ENHANCEMENT_PLACEHOLDER (ANALYTICS.cbl, line 234)
    - Status: Reserved for machine learning integration never implemented
    - Removal effort: 1 week
    - Risk level: MEDIUM - Review with architecture team

6.2 Usage Analytics Summary

Field Usage Frequency:

- High usage (daily): 89 fields
- Medium usage (weekly): 124 fields

- Low usage (monthly): 67 fields
- Unused: 67 fields

Modification Patterns:

- Frequently modified: 45 fields
- Occasionally modified: 78 fields
- Rarely modified: 156 fields
- Never modified: 68 fields

Cross-Program Dependencies:

- Shared across multiple programs: 123 fields
- Program-specific: 224 fields

6.3 Optimization Recommendations

Priority Actions:

1. **Field Consolidation**
   - Action: Merge similar fields like CUSTOMER_ID and CUST_ID
   - Impact: Medium improvement in maintainability
   - Effort: 2 weeks development time
   - Affected programs: 8
2. **Remove Obsolete Fields**
   - Action: Remove 12 obsolete fields identified as safe
   - Impact: Low risk, reduces code bloat
   - Effort: 1 week development time
   - Affected programs: 5
3. **Static Data Externalization**
   - Action: Move static values to configuration table
   - Impact: High improvement in maintenance flexibility
   - Effort: 3 weeks development time
   - Affected programs: 15

6.4 Executive Summary Dashboard

Overall System Health:

- Total fields analyzed: 347
- Field utilization rate: 81%

- Optimization potential: Medium-High
- Technical debt level: Moderate
- Compliance status: Compliant

Key Findings:

- 19% of fields are unused or obsolete - significant cleanup opportunity
- 35% code duplication found across trading programs
- 8 high-priority optimization recommendations identified
- Complete data lineage established for regulatory compliance

Business Impact Projections:

- Maintenance reduction potential: 40%
- Performance improvement estimate: 15-25%
- Compliance readiness: Ready for audit
- Modernization priority: Medium urgency

7. 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"

✅ **Comprehensive field categorization analysis**

- Fields from input: 125 fields identified with source tracking
- Fields updated through processing: 156 fields with calculation logic
- Fields unused and static: 66 fields marked for optimization

✅ **Comparison and optimization reports**

- Similar file analysis with 85% code similarity detection
- Duplicate function identification across 3 trading programs
- 40% maintenance reduction potential through consolidation

8. Sample Data Context: PB 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
```

9. Individual Agent Explanations

Vector Index Agent

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

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.

Example: For a customer security purchase:

1. **CUSTOMER-ID** enters via online trading platform
2. Flows through `VALIDATE.cbl` for user validation 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.

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."

Comparator Agent

Purpose: Identifies similarities, duplications, and optimization opportunities across similar files.

Example: Analyzes multiple trading programs:

- Finds 85% code similarity between SECTRD01, SECTRD02, and SECTRD03
- Identifies 12 duplicate validation functions across programs
- Recommends consolidation to reduce maintenance burden by 40%
- Discovers unused legacy fields that can be safely removed

Documentation Agent

Purpose: Creates human-readable documentation explaining system functionality.

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.

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..."

10. 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)
   - **Comparator Agent**: Identifies similar order processing logic across different trading programs
   - **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
    - Field analysis categorizes all data elements for regulatory reporting
5. **Final Output**:
    - **Fields from input**: Customer ID, Trade Amount, Security Code (from user interface)
    - **Fields updated through processing**: Commission Amount, Net Settlement, Risk Score (calculated)
    - **Fields unused and static**: Legacy account types, obsolete status codes (optimization targets)

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

11. 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
- **comparison_results**: Similar file analysis and optimization recommendations
- **field_classifications**: Input/processed/unused field categorizations

Field Analysis Engine

The system maintains a comprehensive field registry that tracks:

- **Source identification**: Where each field originates (user input, external systems, calculations)
- **Processing lineage**: How fields are transformed through business logic
- **Usage patterns**: Frequency and context of field utilization
- **Optimization opportunities**: Unused, duplicate, or obsolete field identification

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