Multi-Agent Transaction Query System

Technical Documentation

Table of Contents

Executive Summary

The Multi-Agent Transaction Query System is a sophisticated LangGraph-based solution designed to handle customer queries about financial transactions. The system combines multiple specialized agents to provide accurate, policy-compliant responses while handling edge cases gracefully.

Key Features

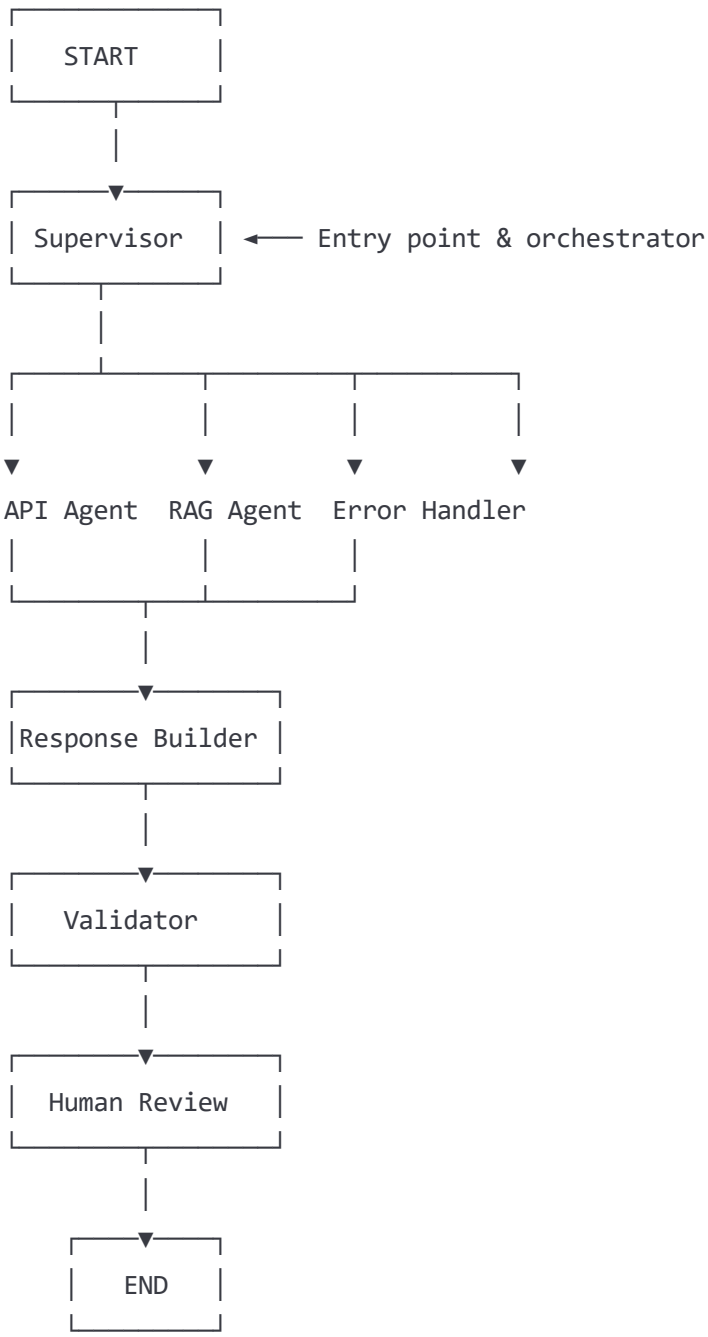
- **Intelligent Query Routing**: Automatically classifies queries and routes to appropriate agents
- **Multi-Source Data Integration**: Combines transaction API data with policy documents
- **Business Rule Validation**: Ensures all responses comply with company policies
- **Human Escalation**: Critical cases are automatically escalated to human agents
- **Error Recovery**: Robust error handling with fallback strategies

---

System Architecture Overview

Core Components

The system consists of 7 specialized agents working together:



Component Details

1. Supervisor Agent

Purpose

The Supervisor is the intelligent orchestrator that analyzes incoming queries and determines the optimal processing path.

Key Functions

- **Query Classification**: Categorizes queries into types (status_check, cancellation, policy_question, etc.)
- **Entity Extraction**: Identifies transaction IDs using regex patterns
- **Routing Decision**: Determines which agents should handle the query

Input Processing

```python
Input: "I need to cancel transaction TPBF000115933785"

Output:
{
    "query_type": "cancellation",
    "transaction_ids": ["TPBF000115933785"],
    "routing_plan": ["api_agent", "rag_agent", "response_builder", "validator"]
}
```

Routing Logic

- **Status Check** → API Agent only
- **Cancellation/Reversal** → API Agent → RAG Agent (for procedure)
- **Policy Question** → RAG Agent only
- **TAT Inquiry** → API Agent → RAG Agent (for TAT guidelines)
- **Unknown/Ambiguous** → RAG Agent (safe default)

---

2. API Agent

Purpose

Retrieves transaction data from the backend API system.

Key Functions

- **Batch Processing**: Handles multiple transaction IDs in a single query
- **Retry Logic**: Implements exponential backoff for failed requests
- **Error Handling**: Gracefully handles missing or invalid transaction IDs

API Integration

```
python

Endpoints used:
- GET /api/transactions/{transaction_id}
- POST /api/transactions/query

Response format:
{
    "transaction_id": "TPBF000115933785",
    "status": "Remit Success",
    "response_code": "3000",
    "modified_on": "2025-01-02T14:30:00Z",
    "formatted_response": "Transaction completed successfully..."
}
```

Error Scenarios

- Transaction not found → Continue with available data
- API timeout → Retry up to 3 times
- Complete API failure → Route to Error Handler

---

3. RAG Agent

Purpose

Searches policy documents for relevant procedures and guidelines.

Key Functions

- **Categorized Search**: Documents are pre-categorized for efficient retrieval
- **Context Enhancement**: Uses transaction status from API to improve search
- **Query Refinement**: Automatically improves search queries if initial results are poor

Document Categories

1. **Cancellation Policies**: Rules for canceling/reversing transactions
2. **TAT Guidelines**: Turn-around time expectations by transaction type
3. **Status Explanations**: Detailed meanings of each status code
4. **KYC Procedures**: Compliance and verification requirements
5. **General Policies**: Other business rules and procedures

Search Strategy

```python
Query: "How to cancel a pending transaction?"
Enhanced queries:
- "cancellation procedure pending status"
- "reverse transaction CREDIT_PENDING"
- "cancel payment awaiting processing"
```

---

4. Response Builder

Purpose

Synthesizes information from multiple sources into a coherent response.

Key Functions

- **Template Selection**: Chooses appropriate response template based on query type
- **Data Integration**: Combines API data with policy information
- **Tone Consistency**: Maintains professional, helpful tone

Synthesis Rules

1. **Prioritize Official Data**: Use API responses when available
2. **Enhance with Context**: Add relevant policy information
3. **Handle Missing Data**: Provide helpful alternatives when data is unavailable
4. **Maintain Clarity**: Keep responses concise and actionable

Example Synthesis

```
API Data: Transaction status = "Remit Success"
RAG Data: "Completed transactions cannot be cancelled"
Query Type: Cancellation request

Synthesized Response:
"I've checked transaction TPBF000115933785 and found that it has been
successfully completed (Remit Success) on 2025-01-02 at 14:30 GMT.

Unfortunately, according to our policies, transactions that have been
successfully delivered to the beneficiary cannot be cancelled. The funds
have already been credited to the recipient's account.

If you believe this transaction was unauthorized, please contact our
fraud department immediately at [fraud support number]."
```

---

5. Validator Agent

Purpose

Ensures responses meet quality standards and comply with business rules.

Validation Checks

1. **Accuracy**: Information matches source data
2. **Completeness**: All aspects of query addressed
3. **Compliance**: No business rule violations
4. **Clarity**: Response is easy to understand
5. **Safety**: No potential for customer harm

Critical Validation Rules

- ❌ Cannot cancel "Remit Success" transactions
- ❌ Cannot provide KYC details without compliance approval
- ⚠️ TAT inquiries must reference official timelines
- ⚠️ Failed transactions must include next steps
- ✅ Status inquiries can be answered directly

Confidence Scoring

```python
Base confidence: 1.0
- No API data for transaction query: -0.3
- No RAG data for policy query: -0.3
- Conflicting information: -0.4
- Missing critical details: -0.2

If confidence < 0.7: Escalate to human review
```

---

6. Human Review Agent

Purpose

Handles cases requiring human judgment or intervention.

Escalation Triggers

- Validation confidence < 0.7
- Cancellation of completed transactions

- KYC/Compliance issues
- Customer disputes
- System errors beyond recovery

Human Interface

```
==========================================
HUMAN REVIEW REQUIRED
==========================================
Reason: Attempting to cancel completed transaction
Query: "Cancel TPBF000115933785 urgent"
Transaction Status: Remit Success
Draft Response: [Generated response]
Suggested Action: Explain policy and offer alternatives
Reference ID: session_12345
==========================================
```

---

7. Error Handler

Purpose

Manages errors and implements recovery strategies.

Error Recovery Strategies

1. **API Failure** → Try SQL as fallback → Route to RAG
2. **RAG Failure** → Continue with API data only
3. **Multiple Failures** → Provide basic response with available info
4. **Cascade Failure** → Escalate to human review

Loop Prevention

- Tracks visit count per agent
- Maximum 3 visits to any agent
- Forces escalation after repeated failures

State Management

State Schema

The system maintains a comprehensive state throughout query processing:

python

```python
{
    # User Input
    "user_query": "Original query text",

    # Query Analysis
    "query_type": "cancellation",
    "transaction_ids": ["TPBF000115933785"],

    # Agent Results
    "api_responses": [{
        "transaction_id": "TPBF000115933785",
        "data": {...},
        "success": true
    }],
    "rag_contexts": [{
        "content": "Policy text...",
        "category": "cancellation_policy",
        "relevance_score": 0.92
    }],

    # Response
    "draft_response": "Synthesized response text",
    "final_response": "Validated response text",
    "confidence_score": 0.85,

    # Control Flow
    "next_agent": "validator",
    "visited_agents": ["supervisor", "api_agent", "rag_agent"],
    "error_count": 0,
    "requires_human": false
}
```

Query Flow Examples

Example 1: Simple Status Check

User Query: "What's the status of transaction TPBF000115933785?"

Flow Sequence

1. **Supervisor**
   - Classifies: `status_check`
   - Extracts: `TPBF000115933785`

- Routes to: `api_agent`

2. **API Agent**
   - Fetches transaction data
   - Status: "Remit Success"
   - Routes to: `response_builder`
3. **Response Builder**
   - Uses API template
   - Confidence: 0.95
   - Routes to: `END`

Final Response:

```
We are pleased to inform you that the transaction with
TerraPay ID: TPBF000115933785 has been successfully
credited to the beneficiary's bank account on
2025-01-02 14:30:00 GMT.

Please notify the sender accordingly.
```

Example 2: Cancellation Request

User Query: "I need to cancel TPBF000115933785 immediately, wrong recipient"

Flow Sequence

1. **Supervisor**
   - Classifies: `cancellation`
   - Extracts: `TPBF000115933785`
   - Routes to: `api_agent`
2. **API Agent**
   - Fetches data
   - Status: "Remit Success"
   - Routes to: `rag_agent`
3. **RAG Agent**
   - Searches cancellation policies
   - Finds: "Cannot cancel completed transactions"
   - Routes to: `response_builder`

4. **Response Builder**
   - Combines API status + policy
   - Confidence: 0.8
   - Routes to: `validator`
5. **Validator**
   - Detects policy violation
   - Sets: `requires_human = true`
   - Routes to: `human_review`
6. **Human Review**
   - Agent reviews case
   - Provides empathetic response with alternatives

Final Response:

```
I understand your concern about sending funds to the wrong recipient.
I've checked transaction TPBF000115933785 and unfortunately, it has
already been successfully completed on 2025-01-02.

According to our policies, we cannot reverse transactions once they've
been delivered. However, here's what you can do:

1. Contact the recipient directly to request a return of funds
2. If you believe this was fraudulent, file a report with our fraud team
3. For future protection, enable transaction confirmation delays

A specialist from our support team will contact you within 2 hours
to assist further. Your reference number is: HR-12345.
```

Example 3: TAT Inquiry

User Query: "Transaction TPLM000115933792 showing CREDIT_DELAYED, is this normal?"

Flow Sequence

1. **Supervisor**
   - Classifies: `tat_inquiry`
   - Extracts: `TPLM000115933792`
   - Routes to: `api_agent`
2. **API Agent**
   - Status: "CREDIT_DELAYED"

- Transaction time: 3 hours ago
- Routes to: `rag_agent`

3. **RAG Agent**
   - Searches TAT guidelines
   - Finds: "Bank transfers: 2-4 hours"
   - Routes to: `response_builder`
4. **Response Builder**
   - Notes transaction within TAT
   - Confidence: 0.9
   - Routes to: `END`

Final Response:

```
I've checked transaction TPLM000115933792 which is currently showing
as CREDIT_DELAYED. The transaction was initiated 3 hours ago.

This is completely normal. According to our processing guidelines:
- Bank transfers typically take 2-4 hours
- Your transaction is still within the expected timeframe

The status will update to "Remit Success" once the funds are credited
to the beneficiary's account. You can track real-time updates in the
ENGAGE portal.

If the status doesn't change within 4 hours, please contact us again
and we'll investigate immediately.
```

Example 4: Policy Question

User Query: "What documents are needed for KYC verification?"

Flow Sequence

1. **Supervisor**
   - Classifies: `policy_question`
   - No transaction IDs
   - Routes to: `rag_agent`
2. **RAG Agent**
   - Searches KYC procedures

- Finds detailed requirements
- Routes to: `response_builder`

3. **Response Builder**
   - Formats policy information
   - Confidence: 0.92
   - Routes to: `END`

Final Response:

```
For KYC verification, you'll need to provide:

1. **Identity Proof** (any one):
   - Passport
   - National ID card
   - Driver's license

2. **Address Proof** (any one):
   - Utility bill (last 3 months)
   - Bank statement
   - Rental agreement

3. **Additional Information**:
   - Purpose of transaction
   - Source of funds (for large amounts)

Please upload clear, colored copies of these documents through
our secure portal. Verification typically takes 24-48 hours.

If your transaction is currently on hold for KYC, providing these
documents will allow us to process it immediately.
```

Example 5: Error Recovery

User Query: "Check status of TPBF000115933785"

Flow with API Failure

1. **Supervisor** → Routes to `api_agent`
2. **API Agent** → Timeout error → Routes to `error_handler`
3. **Error Handler** → Retry API → Still fails → Routes to `rag_agent`
4. **RAG Agent** → Provides general status information
5. **Response Builder** → Creates response with available info

Final Response:

```
I'm experiencing some technical difficulties accessing the specific
details for transaction TPBF000115933785.

Here's what I can help you with:
- You can check your transaction status in the ENGAGE portal
- Most transactions complete within:
  * 5-10 minutes for wallet transfers
  * 2-4 hours for bank transfers

If your transaction is urgent, please:
1. Try again in a few minutes
2. Contact our support line at [phone number]
3. Reference transaction ID: TPBF000115933785

I apologize for the inconvenience.
```

Business Rules & Edge Cases

Critical Business Rules

1. Transaction Status Rules

Status	Can Cancel?	Can Modify?	TAT
Remit Success	❌ No	❌ No	N/A
Remit Failed	✅ Yes	❌ No	N/A
CREDIT_PENDING	✅ Yes*	❌ No	Check TAT
KYC_PENDING	⚠️ Compliance	❌ No	5 days
ON_HOLD	✅ Yes	❌ No	Varies

*Within time limits

2. Edge Case Handling

Multiple Transaction IDs

- Process each ID separately
- Aggregate results in response
- Show status for each transaction

Invalid Transaction IDs

- Skip invalid IDs
- Process valid ones
- Notify user about invalid formats

Mixed Valid/Invalid

- Process what's possible
- Clear reporting on failures
- Helpful suggestions for invalid IDs

No Transaction ID

- Default to policy/help information
- Suggest how to find transaction ID
- Offer general assistance

Ambiguous Queries

- Start with RAG for safety
- Provide comprehensive help
- Offer to clarify specifics

Implementation Guidelines

1. Development Phases

Phase 1: Core Infrastructure (Week 1)

- Set up LangGraph workflow
- Implement state management
- Create basic supervisor logic
- Build API integration

Phase 2: Agent Implementation (Week 2)

- Develop API Agent with retry logic
- Build RAG system with categorization
- Create Response Builder
- Implement Validator

Phase 3: Advanced Features (Week 3)

- Add Error Handler
- Implement Human Review flow
- Enhance routing intelligence
- Add conversation memory

Phase 4: Testing & Optimization (Week 4)

- End-to-end testing
- Performance optimization
- Edge case validation
- Load testing

2. Technology Stack

Required Components

- **LangGraph**: Workflow orchestration
- **LangChain**: LLM integration
- **FastAPI**: REST API backend
- **PostgreSQL/SQLite**: Transaction database
- **FAISS**: Vector store for RAG
- **OpenAI/Claude**: LLM for intelligence

Optional Enhancements

- **Redis**: Caching layer
- **Celery**: Async task processing
- **Elasticsearch**: Advanced search
- **Grafana**: Monitoring dashboard

3. Monitoring & Metrics

Key Performance Indicators

- Query resolution time (target: <3 seconds)
- Human escalation rate (target: <10%)
- Response accuracy (target: >95%)
- System uptime (target: 99.9%)

Monitoring Points

- Agent processing times

- API response times
- RAG search quality
- Error rates by type
- Confidence score distribution

4. Security Considerations

Data Protection

- Encrypt transaction IDs in logs
- Mask sensitive information
- Implement rate limiting
- Add authentication layers

Audit Trail

- Log all agent decisions
- Track human interventions
- Record state transitions
- Monitor unusual patterns

Conclusion

The Multi-Agent Transaction Query System provides a robust, scalable solution for handling complex customer queries. By combining specialized agents with intelligent routing and comprehensive error handling, the system delivers accurate, policy-compliant responses while maintaining high availability and performance.

The modular architecture allows for easy extension and modification as business requirements evolve, while the built-in validation and human escalation ensure safety and compliance in all scenarios.

Appendix: Quick Reference

Transaction ID Patterns

- TPBF + 12 digits
- TPSQ + 12 digits
- TPLM + 12 digits
- TPWU + 12 digits
- TPCE + 12 digits

Response Codes

- 3000: Remit Success
- 4000: Remit Failed
- 5001: KYC_VERIFICATION_PENDING
- 5002: CREDIT_PENDING
- 5003: CREDIT_DELAYED
- 6001: ON HOLD INSUFF FUNDS

Common TAT Guidelines

- Wallet transfers: 5-10 minutes
- Domestic bank: 2-4 hours
- International: 24-48 hours
- With KYC hold: +5 days