SAGE-CODE: Code-Augmented Reasoning on Adaptive Graphs

A Comprehensive Technical Report

Version: 1.0.0
Date: January 2026
Domain: Clinical Intelligence & Data Quality Analytics

Executive Summary

SAGE-CODE (**S**emantic **A**daptive **G**raph **E**ngine with **CODE** augmentation) is a novel hybrid Graph RAG (Retrieval-Augmented Generation) framework designed for clinical trial data intelligence. It combines knowledge graph traversal with dynamic code execution to provide actionable, data-backed insights for clinical trial management and data quality assurance.

The framework addresses the limitations of traditional RAG systems by:
1. **Adaptive Graph Traversal**: Using LLM-guided multi-hop reasoning to navigate complex clinical trial knowledge graphs
2. **Code Augmentation**: Enabling on-the-fly Python code generation and execution for analytical queries
3. **Semantic Selection**: Employing batched LLM scoring for intelligent candidate pruning during graph exploration
4. **Chain-of-Thought Reasoning**: Integrating structured reasoning patterns for decision-making during traversal

---

Table of Contents

---

1. Introduction

1.1 Problem Statement

Clinical trial management generates vast amounts of structured and semi-structured data across multiple domains:
- **Safety Events (eSAE)**: Serious Adverse Events requiring regulatory reporting
- **Data Discrepancies (EDRR)**: Open issues and data quality problems
- **Medical Coding (MedDRA/WHODD)**: Standardized coding for adverse events and medications
- **Visit Projections**: Patient visit scheduling and compliance tracking
- **Missing Data**: Form completion gaps and data entry delays

Traditional analytics approaches struggle with:
- Complex entity relationships spanning multiple data sources
- Need for both relational queries and analytical computations
- Requirement for explainable, data-backed decision support
- Integration of natural language queries with structured data analysis

1.2 SAGE-CODE Solution

SAGE-CODE addresses these challenges through a hybrid architecture that:

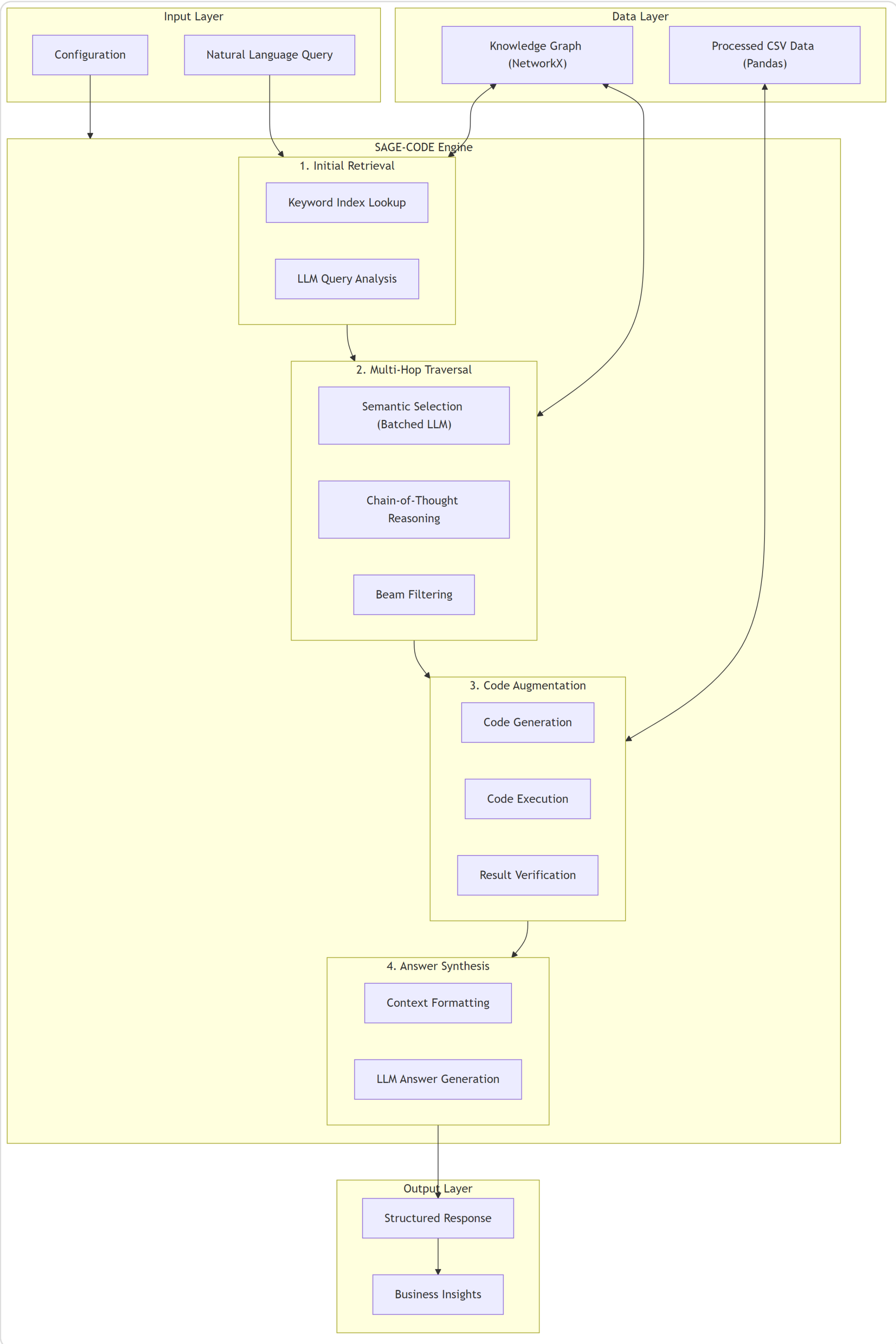
1. **Unifies Data**: Constructs a comprehensive knowledge graph from disparate CSV data sources
2. **Enables Natural Language Queries**: Allows clinical teams to ask questions in plain English
3. **Provides Grounded Answers**: Generates responses backed by actual data, not hallucinations
4. **Supports Analytics**: Executes Python code for complex aggregations and statistics

5. **Explains Reasoning**: Uses Chain-of-Thought patterns to show how conclusions were reached

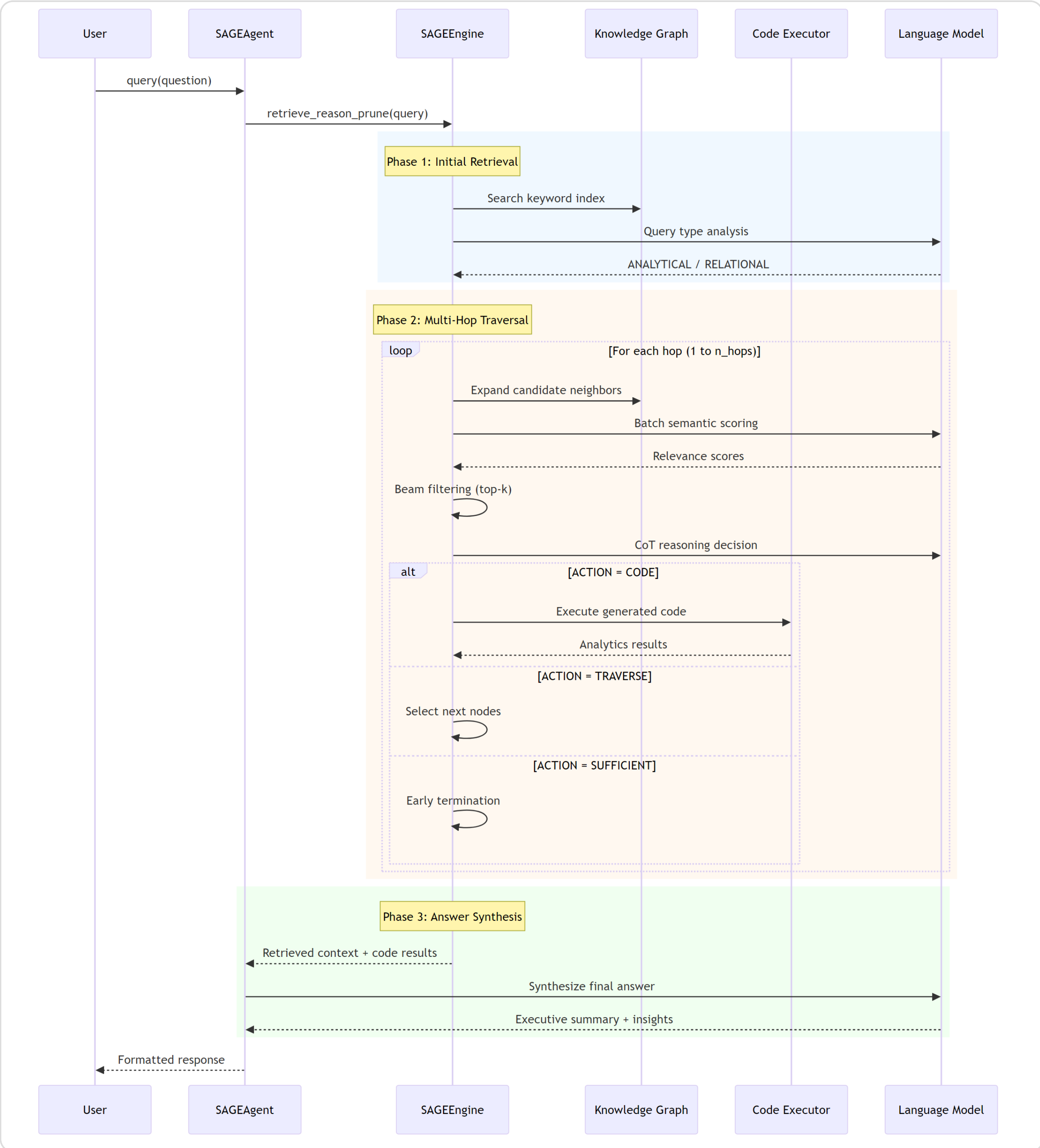
---

2. Architecture Overview

2.1 High-Level System Architecture

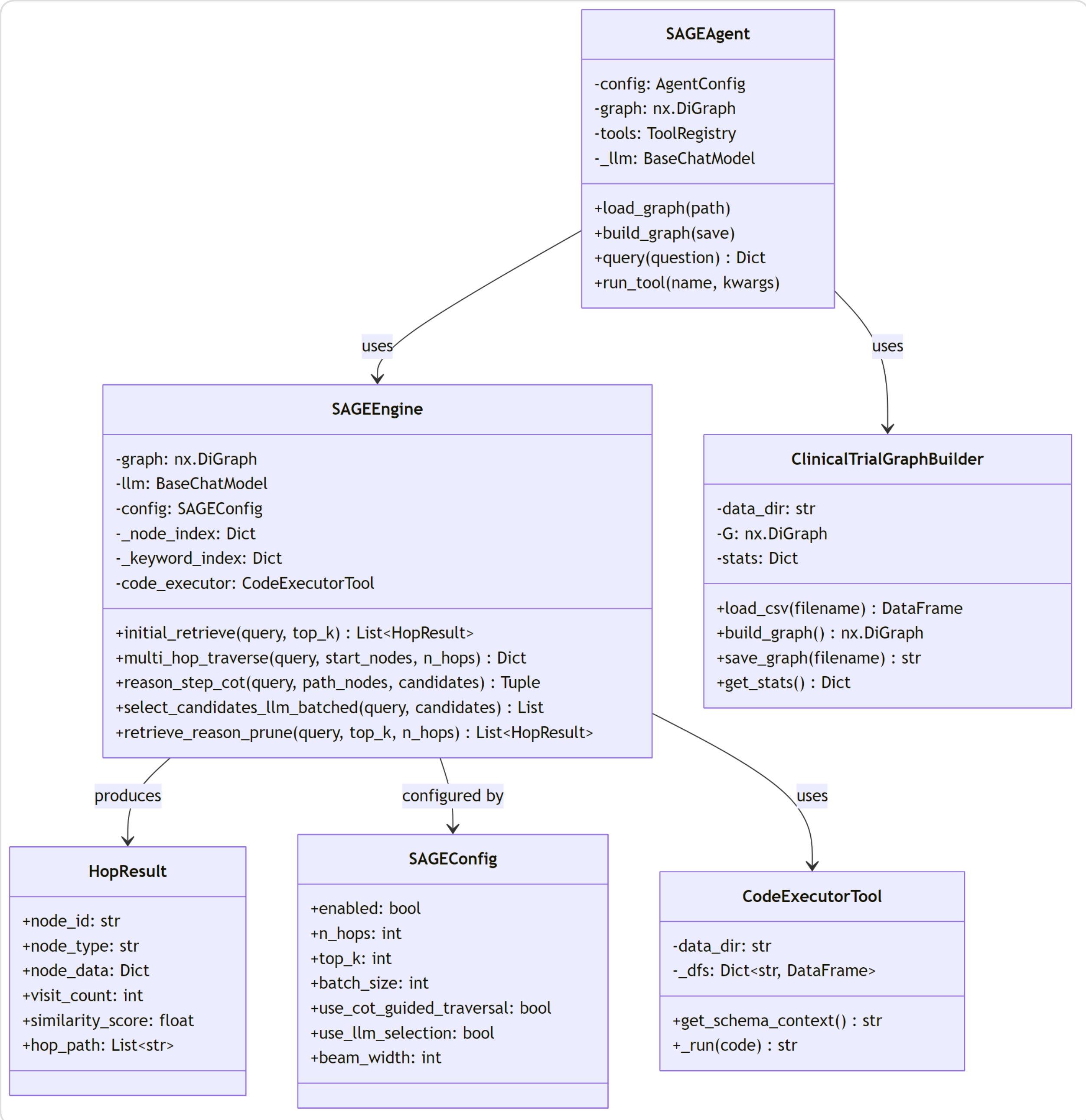

Input Layer
Configuration
Natural Language Query
Data Layer
Knowledge Graph
(NetworkX)
Processed CSV Data
(Pandas)
SAGE-CODE Engine
1. Initial Retrieval
Keyword Index Lookup
LLM Query Analysis
2. Multi-Hop Traversal
Semantic Selection
(Batched LLM)
Chain-of-Thought
Reasoning
Beam Filtering
3. Code Augmentation
Code Generation
Code Execution
Result Verification
4. Answer Synthesis
Context Formatting
LLM Answer Generation
Output Layer
Structured Response
Business Insights

2.2 Component Interaction Flow



3. Core Components

3.1 Component Architecture



3.2 SAGEAgent

The `SAGEAgent` class serves as the primary interface for interacting with the SAGE-CODE system:

Key Responsibilities:
- Loading or building the knowledge graph
- Managing the tool registry

- Orchestrating queries through the SAGE engine
- Synthesizing final answers using LLM

Initialization Flow:
1. Load configuration (LLM settings, graph paths, SAGE parameters)
2. Load or build knowledge graph from CSV data
3. Initialize tool registry with graph and code execution tools
4. Set up LLM connection (supports Groq, Google, OpenAI providers)

3.3 SAGEEngine

The `SAGEEngine` is the core retrieval-reasoning component:

Key Features:
- **Inverted Keyword Index**: O(1) lookup for initial node retrieval
- **LLM Query Analysis**: Distinguishes ANALYTICAL vs RELATIONAL queries
- **Batched LLM Scoring**: Efficient semantic relevance scoring
- **CoT-Guided Traversal**: Intelligent path selection with reasoning
- **Code Augmentation**: Dynamic code generation and execution

3.4 HopResult

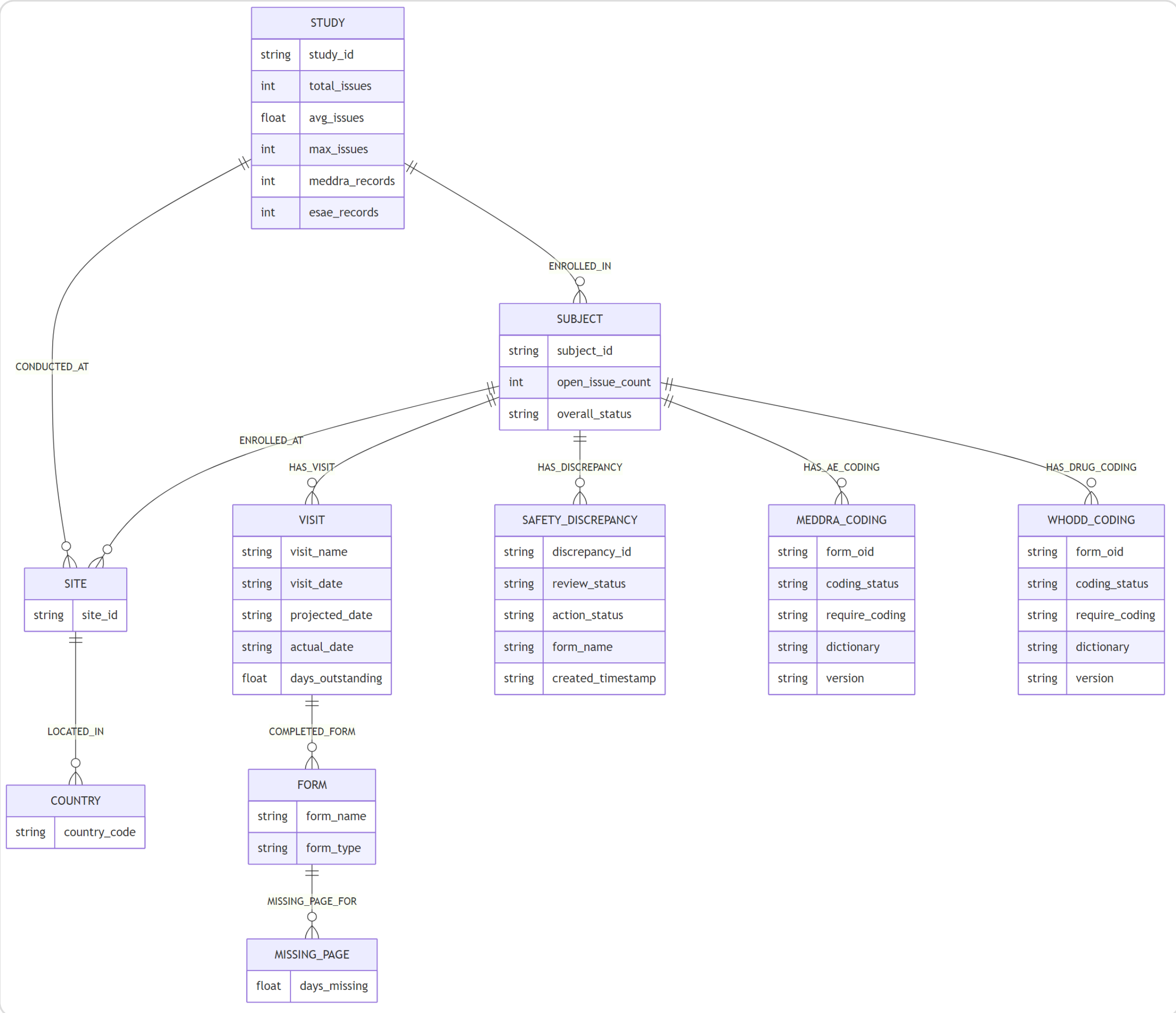
Data class representing a visited node during graph traversal:

```
@dataclass
class HopResult:
    node_id: str           # e.g., "SITE:637"
    node_type: str         # e.g., "Site", "Subject", "SafetyDiscrepancy"
    node_data: Dict        # All node attributes
    visit_count: int       # Times visited during traversal
    similarity_score: float # Relevance to query (0-1)
    hop_path: List[str]    # Path from initial node to this node
```

---

4. Knowledge Graph Construction

4.1 Graph Schema

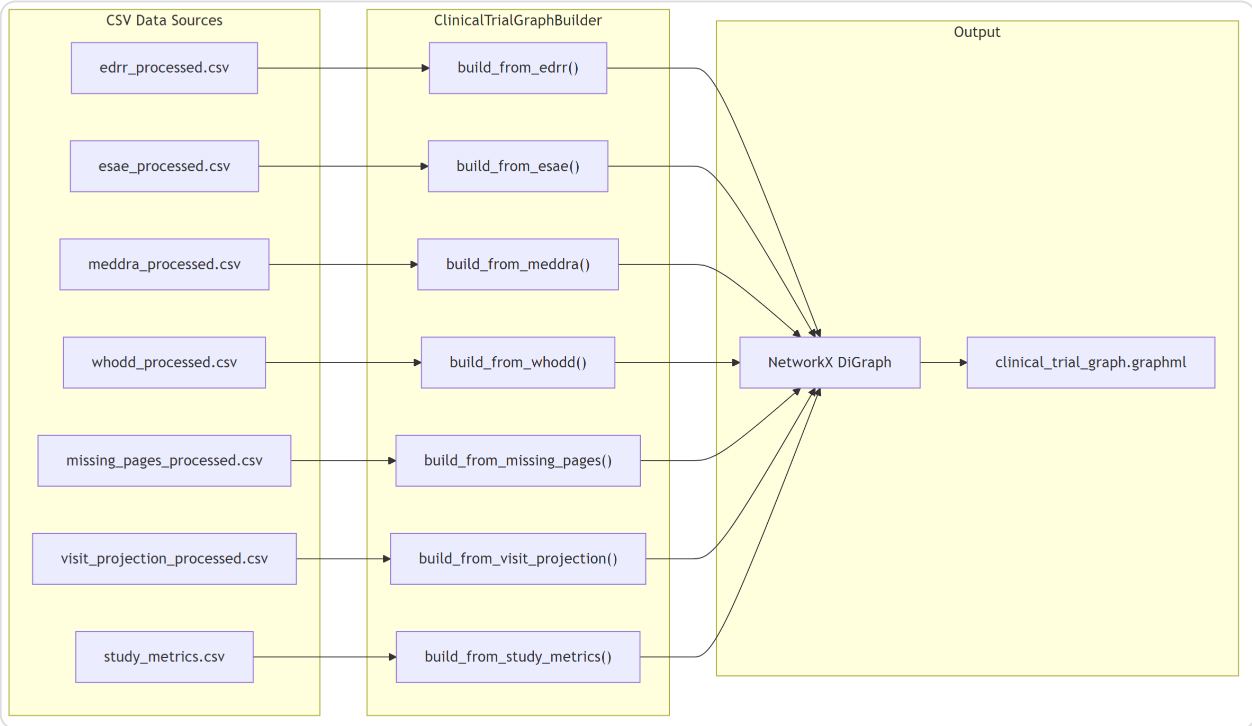


4.2 Data Sources

The `ClinicalTrialGraphBuilder` processes the following CSV files:

Source File	Node Types Created	Edge Types Created
`edrr_processed.csv`	Study, Subject	ENROLLED_IN
`esae_processed.csv`	Study, Country, Site, Subject, SafetyDiscrepancy	LOCATED_IN, CONDUCTED_AT, ENROLLED_IN, ENROLLED_AT, HAS_DISCREPANCY
`meddra_processed.csv`	Study, Subject, MedDRACoding	ENROLLED_IN, HAS_AE_CODING
`whodd_processed.csv`	Study, Subject, WHODDCoding	ENROLLED_IN, HAS_DRUG_CODING
`missing_pages_processed.csv`	Study, Country, Site, Subject, Visit, Form, MissingPage	LOCATED_IN, CONDUCTED_AT, ENROLLED_IN, ENROLLED_AT, HAS_VISIT, COMPLETED_FORM, MISSING_PAGE_FOR
`visit_projection_processed.csv`	Study, Country, Site, Subject, Visit	LOCATED_IN, CONDUCTED_AT, ENROLLED_IN, ENROLLED_AT, HAS_VISIT
`study_metrics.csv`	(Enriches existing Study nodes)	-

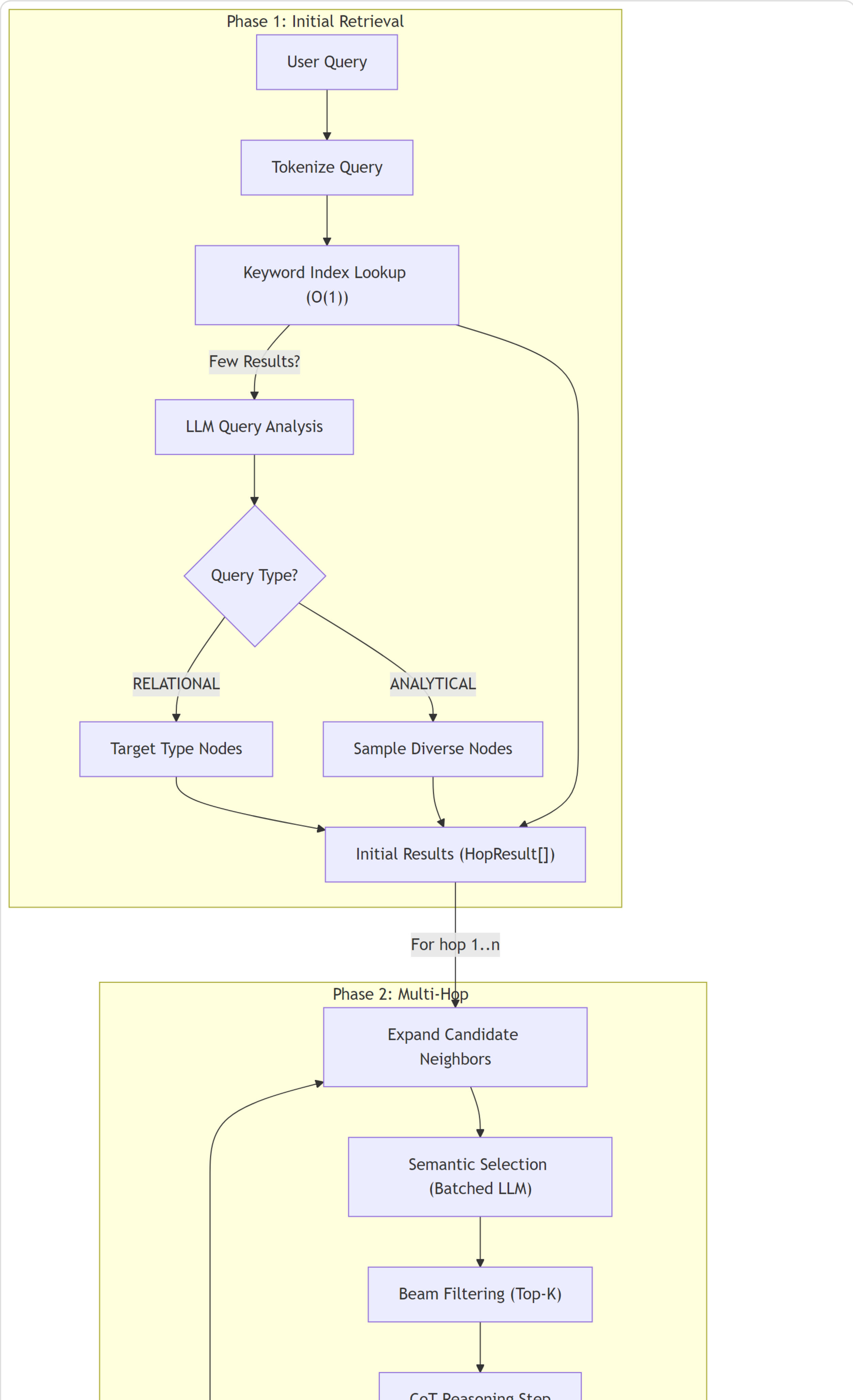
4.3 Graph Construction Process



5. Retrieval-Reasoning Pipeline

5.1 Pipeline Overview

The SAGE-CODE pipeline follows a **Retrieve** → **Reason** → **Prune** pattern:



Phase 1: Initial Retrieval
User Query
Tokenize Query
Keyword Index Lookup (O(1))
Few Results?
LLM Query Analysis
Query Type?
RELATIONAL
ANALYTICAL
Target Type Nodes
Sample Diverse Nodes
Initial Results (HopResult[])
For hop 1..n
Phase 2: Multi-Hop
Expand Candidate Neighbors
Semantic Selection (Batched LLM)
Beam Filtering (Top-K)

5.2 Initial Retrieval

Step 1: Keyword Index Lookup
- Query is tokenized into lowercase words
- Inverted index provides O(1) lookup: `word → [node_id, ...]`
- Nodes are scored by match overlap: `score = matches / query_words`

Step 2: LLM Query Analysis (if insufficient matches)

```
analysis_prompt = """
Analyze this clinical trial query and determine:
1. query_type: "ANALYTICAL" or "RELATIONAL"
2. target_types: List of node types to search
3. key_entities: Specific entity IDs mentioned
"""
```

Step 3: Query Type Handling
- **ANALYTICAL**: Sample diverse nodes for code context
- **RELATIONAL**: Target specific node types for traversal

5.3 Semantic Selection (Batched)

The `select_candidates_llm_batched` function scores candidates using LLM:

```
BATCH_SELECTION_PROMPT = """
Rate relevance (0-10) of each candidate node for the query.
Consider:
1. Explicit matches (keywords)
2. Semantic relationships
3. DATA POTENTIAL: Score HIGHER if the node unlocks computation

Query: "{query}"
Candidates: {candidates_text}

Response: {"scores": {"candidate_index_0": 9, ...}}
"""
```

5.4 Chain-of-Thought Reasoning

The CoT step uses a code-aware prompt that provides:
- Current path context (visited nodes)
- Candidate nodes for traversal
- Available DataFrame schemas

Decision Actions:

Action	Description	Effect
`CODE`	Generate and execute Python code	Analytics query, may terminate early
`TRAVERSE`	Select nodes for next hop	Continue graph exploration
`SUFFICIENT`	Information complete	Terminate traversal

5.5 Helpfulness Scoring

Final results are scored using:

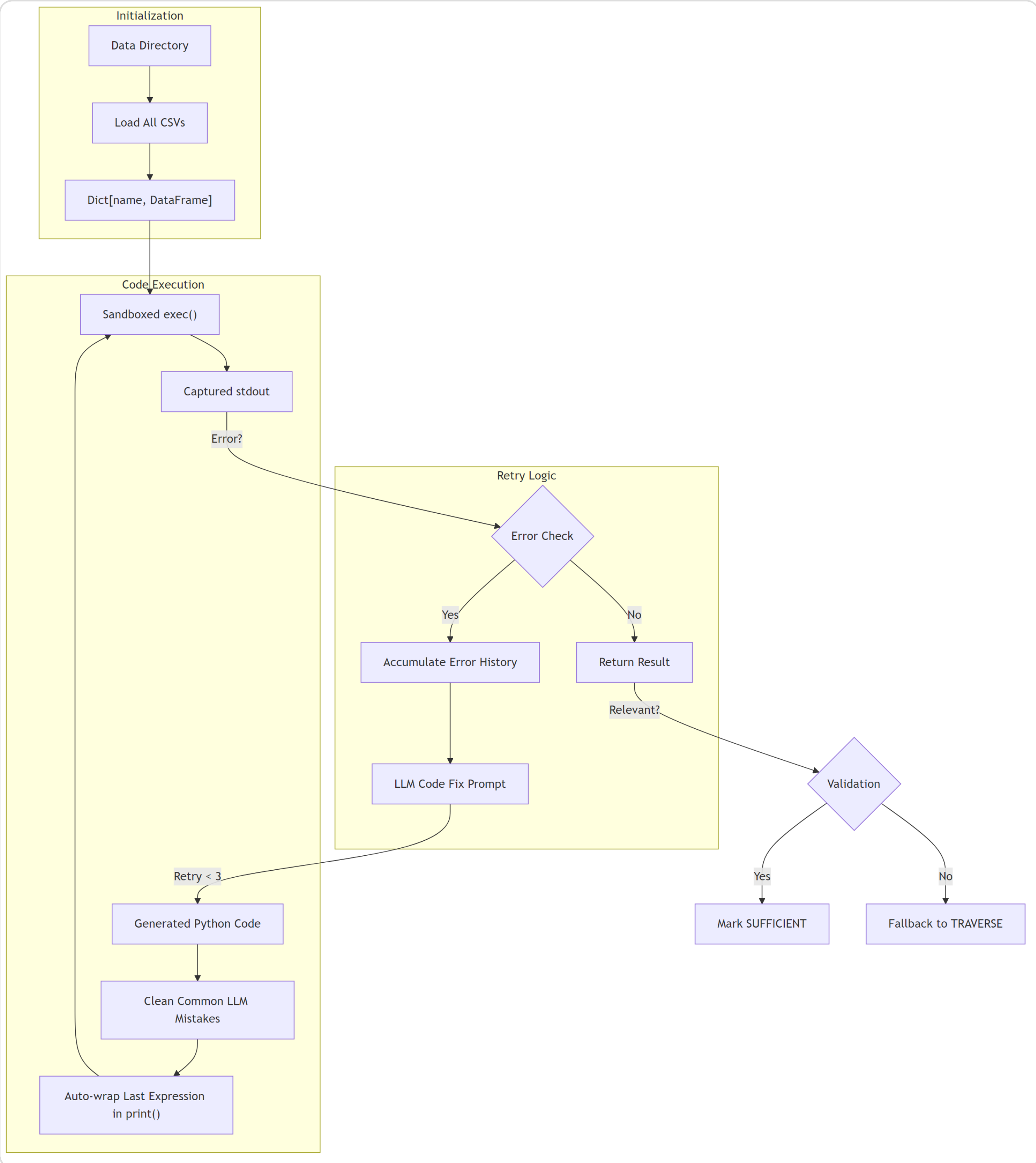
```
helpfulness = similarity_weight * sim_score + (1 - similarity_weight) * importance_score
```

Where:
- `sim_score`: Semantic similarity to query
- `importance_score`: Normalized visit count during traversal

---

6. Code Augmentation System

6.1 Code Executor Architecture



6.2 Available DataFrames

The code executor dynamically loads all CSVs from the processed data directory:

DataFrame Name	Source File	Key Columns
`esae_processed_df`	esae_processed.csv	study_id, site, patient_id, review_status
`missing_pages_processed_df`	missing_pages_processed.csv	study_name, sitenumber, subjectname, no___days_page_missing
`meddra_processed_df`	meddra_processed.csv	study, subject, coding_status
`whodd_processed_df`	whodd_processed.csv	study, subject, coding_status
`visit_projection_processed_df`	visit_projection_processed.csv	study, site, subject, __days_outstanding
`study_metrics_df`	study_metrics.csv	study, total_issues, avg_issues
`edrr_processed_df`	edrr_processed.csv	study, subject, total_open_issue_count_per_subject

6.3 Code Generation Example

Query: "Which sites have the most pending safety reviews?"

Generated Code:

```
result = esae_processed_df[
    esae_processed_df['review_status'] == 'Pending for Review'
].groupby('site').size().sort_values(ascending=False).head(10)
print(result)
```

Output:

```
site
637     45
412     38
...
```

6.4 Self-Healing Code Execution

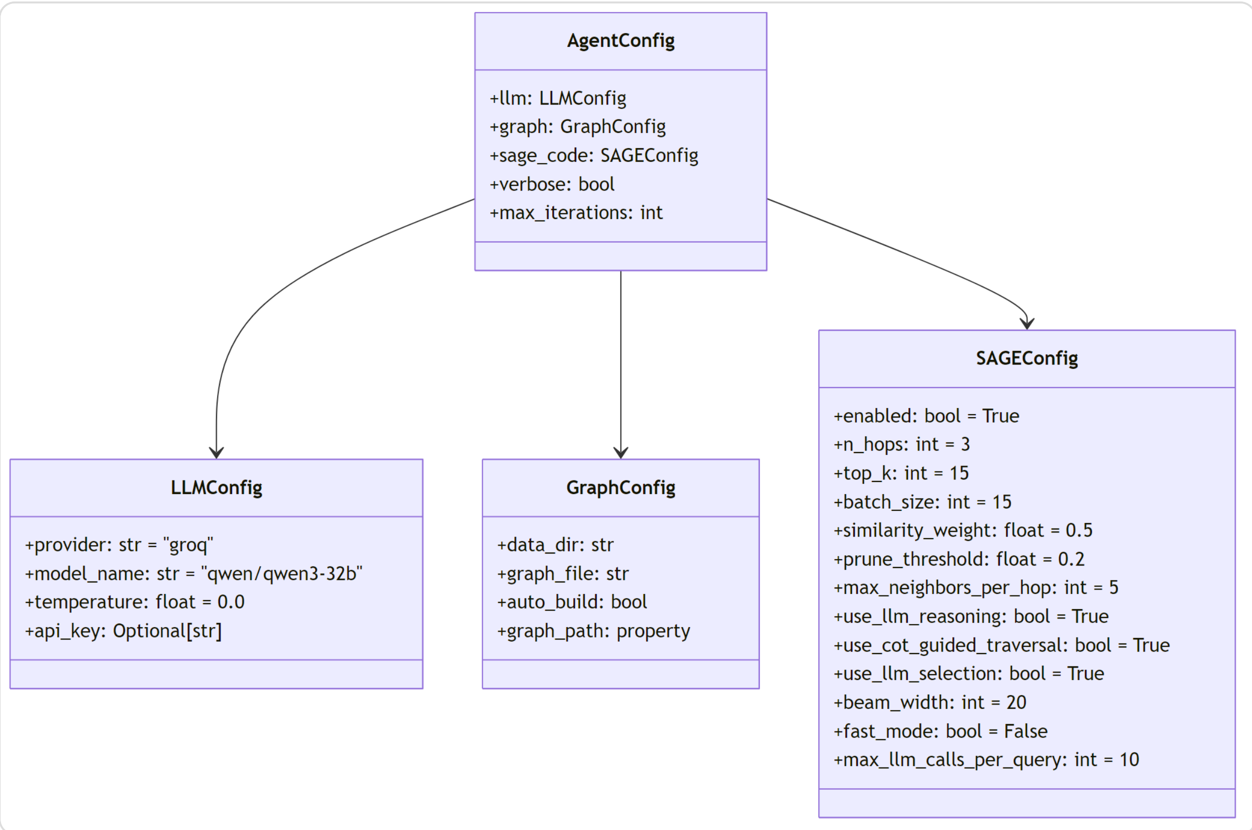
The system implements automatic error recovery:

1. **Error Detection**: Check if output starts with "Error:"
2. **Error Accumulation**: Track all errors across retries
3. **LLM Fix Request**: Provide full error history to avoid repeated mistakes
4. **Maximum Retries**: 3 attempts before falling back to traversal

---

7. Configuration & Customization

7.1 Configuration Classes



7.2 Key Configuration Parameters

Parameter	Default	Description
`n_hops`	3	Maximum traversal depth
`top_k`	15	Number of results to return
`batch_size`	15	LLM batch size for scoring
`beam_width`	20	Candidates kept per hop
`similarity_weight`	0.5	Weight for semantic vs structural scoring
`prune_threshold`	0.2	Minimum helpfulness score
`use_cot_guided_traversal`	True	Enable CoT reasoning
`use_llm_selection`	True	Enable semantic scoring
`fast_mode`	False	Skip LLM reasoning (keyword only)
`skip_multi_hop`	False	Return only initial retrieval

7.3 Performance Modes

Full Mode (Default):
- All LLM features enabled
- Best accuracy for complex queries
- Higher latency and API cost

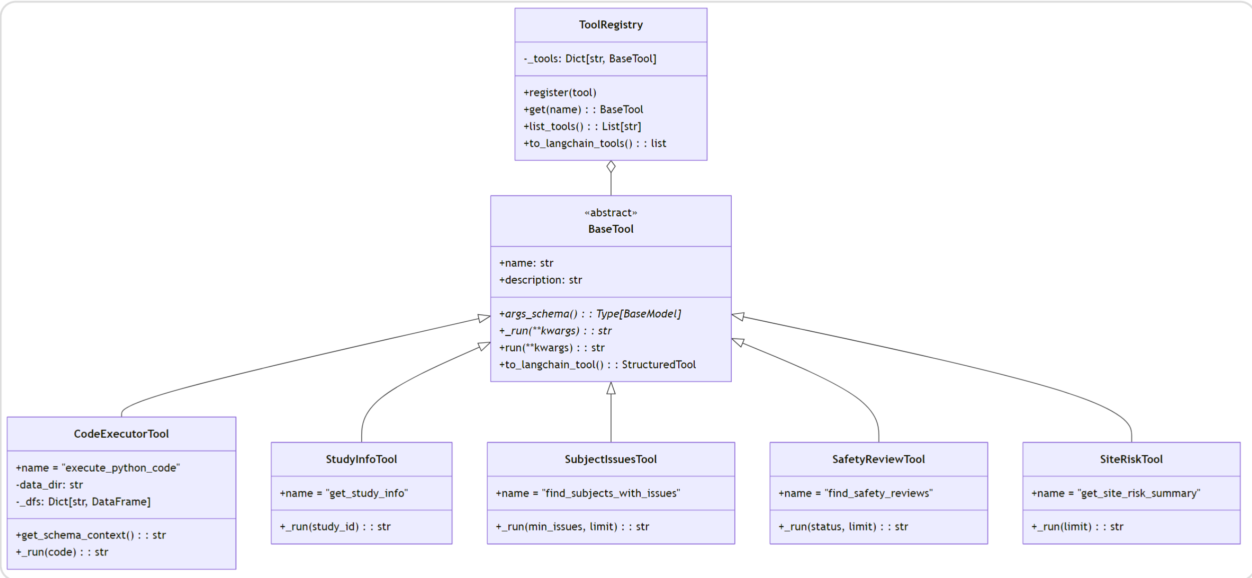
Fast Mode:

```
config = SAGEConfig(
    fast_mode=True,
    skip_multi_hop=True,
    use_llm_reasoning=False
)
```

- Keyword retrieval only
- Minimal latency
- Suitable for simple lookups

8. Tools & Extensions

8.1 Tool Architecture



8.2 Available Tools

Tool Name	Description	Key Parameters
`execute_python_code`	Run pandas code on clinical data	`code: str`
`get_study_info`	Get study information and metrics	`study_id: str`
`find_subjects_with_issues`	Find subjects with open issues	`min_issues: int` , `limit: int`
`find_safety_reviews`	Find safety discrepancies by status	`status: str` , `limit: int`
`get_safety_reviews_by_site`	Aggregate safety reviews by site	`status: str` , `limit: int`
`find_missing_pages`	Find aged missing pages	`min_days: float` , `limit: int`
`get_site_risk_summary`	Get risk summary by site	`limit: int`

8.3 DQI Analytics Tools (Extension)

Additional analytics capabilities:

Tool	Description
`calculate_dqi`	Calculate Data Quality Index score
`explain_dqi`	LLM-generated DQI analysis
`rank_sites`	Rank sites by metrics
`benchmark_site`	Compare site against peers
`identify_underperformers`	Find sites needing attention

9. Use Cases & Applications

9.1 Clinical Trial Data Quality

Example Query: "Is the current data snapshot clean enough for interim analysis?"

SAGE-CODE Response:

```
### Executive Summary
The data shows moderate readiness with 23% of sites having pending safety reviews.

### Key Findings
| Metric | Value | Status |
|--------|-------|--------|
| Open Issues | 1,247 | ⚠ Elevated |
| Pending Reviews | 456 | Action Required |
| Missing Pages >30d | 89 | Needs Resolution |

### Recommendations
1. Prioritize Site 637 for data cleanup (45 pending reviews)
2. Resolve missing pages before lock
3. Complete MedDRA coding for 12 outstanding AEs
```

9.2 Site Performance Analysis

Example Query: "Which sites have the most data quality issues?"

Process:
1. CoT decides: `ACTION = CODE`
2. Generates aggregation code
3. Executes against `esae_processed_df`
4. Returns ranked site list with issue counts

9.3 Subject Safety Tracking

Example Query: "Show me subjects with more than 5 open issues enrolled at Site 412"

Process:
1. Initial retrieval finds Site 412 node
2. Traversal explores ENROLLED_AT edges
3. Filters subjects by issue count
4. Returns subject list with details

10. Technical Specifications

10.1 Dependencies

Package	Version	Purpose
`networkx`	≥2.8	Graph data structure
`pandas`	≥1.5	DataFrame operations
`pydantic`	≥2.0	Data validation
`langchain-core`	≥0.1	LLM abstraction
`langchain-groq`	≥0.1	Groq LLM integration
`langchain-google-genai`	≥0.1	Google LLM integration
`langchain-openai`	≥0.1	OpenAI LLM integration

10.2 LLM Providers

Provider	Model	Use Case
Groq	qwen/qwen3-32b	Default, fast inference
Google	gemini-1.5-pro	Multi-modal, large context
OpenAI	gpt-4o	High accuracy

10.3 Performance Characteristics

Metric	Typical Value
Initial Retrieval	<100ms
Per-Hop LLM Call	500-2000ms
Code Execution	50-500ms
Total Query (3 hops)	3-10 seconds
LLM Calls per Query	3-12

10.4 Graph Statistics (Typical)

Metric	Value
Total Nodes	200,000+
Total Edges	500,000+
Node Types	10+
Edge Types	12+

Appendix A: LLM Prompts

A.1 SAGE Agent System Prompt

```
You are a Senior Clinical Trial Consultant AI. Your role is to provide
clear, actionable business insights derived from complex data.

## CRITICAL RESPONSE GUIDELINES
1. Human-Centric & Professional: Write for business stakeholders
2. No Technical Details in Output
3. Data-Backed Insights: Always include specific numbers
4. Action-Oriented: Conclude with specific next steps
```

A.2 Code-Augmented CoT Prompt

```
You are an intelligent clinical trial data analyst with Python Pandas capabilities.

Goal: Answer the user's query using the available dataframes.
PREFER CODE over traversal for analytical questions.

DECISION RULES:
1. Use CODE if the query asks for: counts, aggregations, rankings, statistics
2. Use TRAVERSE only if you need to find specific entity IDs or relationships
3. Use SUFFICIENT only if the Current Knowledge already contains the answer
```

A.3 Batch Selection Prompt

```
Rate relevance (0-10) of each candidate node for the query.
Consider:
1. Explicit matches (keywords)
2. Semantic relationships
3. DATA POTENTIAL: Score HIGHER if the node unlocks computation
```

Appendix B: File Structure

```
sage_code/
├── __init__.py          # Package exports
├── agent.py             # SAGEAgent class
├── engine.py            # SAGEEngine (core algorithm)
├── graph_builder.py     # ClinicalTrialGraphBuilder
├── config.py            # Configuration dataclasses
├── models.py            # HopResult dataclass
├── prompts.py           # LLM prompt templates
└── tools/
    ├── __init__.py      # Tools exports
    ├── base_tool.py     # BaseTool, ToolRegistry
    ├── code_executor.py # CodeExecutorTool
    ├── graph_tools.py   # Graph query tools
    └── dqi_analytics_tools.py  # DQI extension tools
```

References

1. **Graph RAG**: Lewis, P. et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.
2. **Chain-of-Thought Prompting**: Wei, J. et al. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.
3. **Knowledge Graphs for Clinical Trials**: Various industry implementations.

Document generated for Novartis Clinical Intelligence Platform
SAGE-CODE Framework v1.0.0