TRIALS: Text-to-SQL with Ranked Iterative Agent Learning and Selection

A Comprehensive Technical Report

Version: 1.0.0
Date: January 2026
Domain: Clinical Trial Data Intelligence

Executive Summary

TRIALS (**T**ext-to-SQL with **R**anked **I**terative **A**gent **L**earning and **S**election) is a sophisticated multi-agent Text-to-SQL system designed for querying clinical trial data using natural language. The system transforms natural language questions into accurate SQL queries and provides human-readable explanations of results.

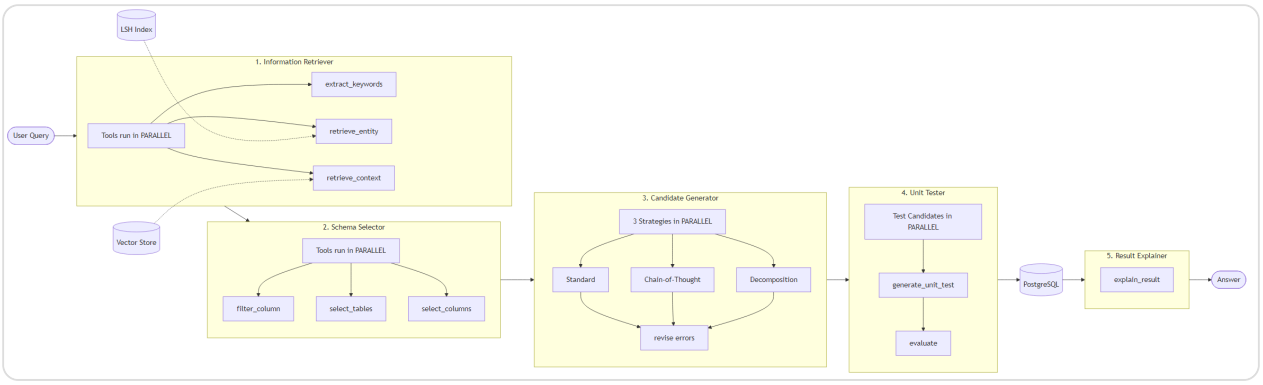
The framework employs an ensemble approach with **5 specialized AI agents** working in coordination:
1. **Information Retriever** - Gathers context using LSH and vector search
2. **Schema Selector** - Reduces schema to relevant tables/columns
3. **Candidate Generator** - Produces multiple SQL candidates using parallel strategies
4. **Unit Tester** - Selects best candidate through generated tests
5. **Result Explainer** - Converts results to natural language answers

Key Innovations:
- Multi-strategy SQL generation (Standard, Chain-of-Thought, Decomposition)
- Self-healing queries with automatic error detection and revision
- Unit test-based candidate selection for robustness
- LSH-based entity retrieval for fast approximate string matching
- Parallel execution at multiple levels for optimal performance

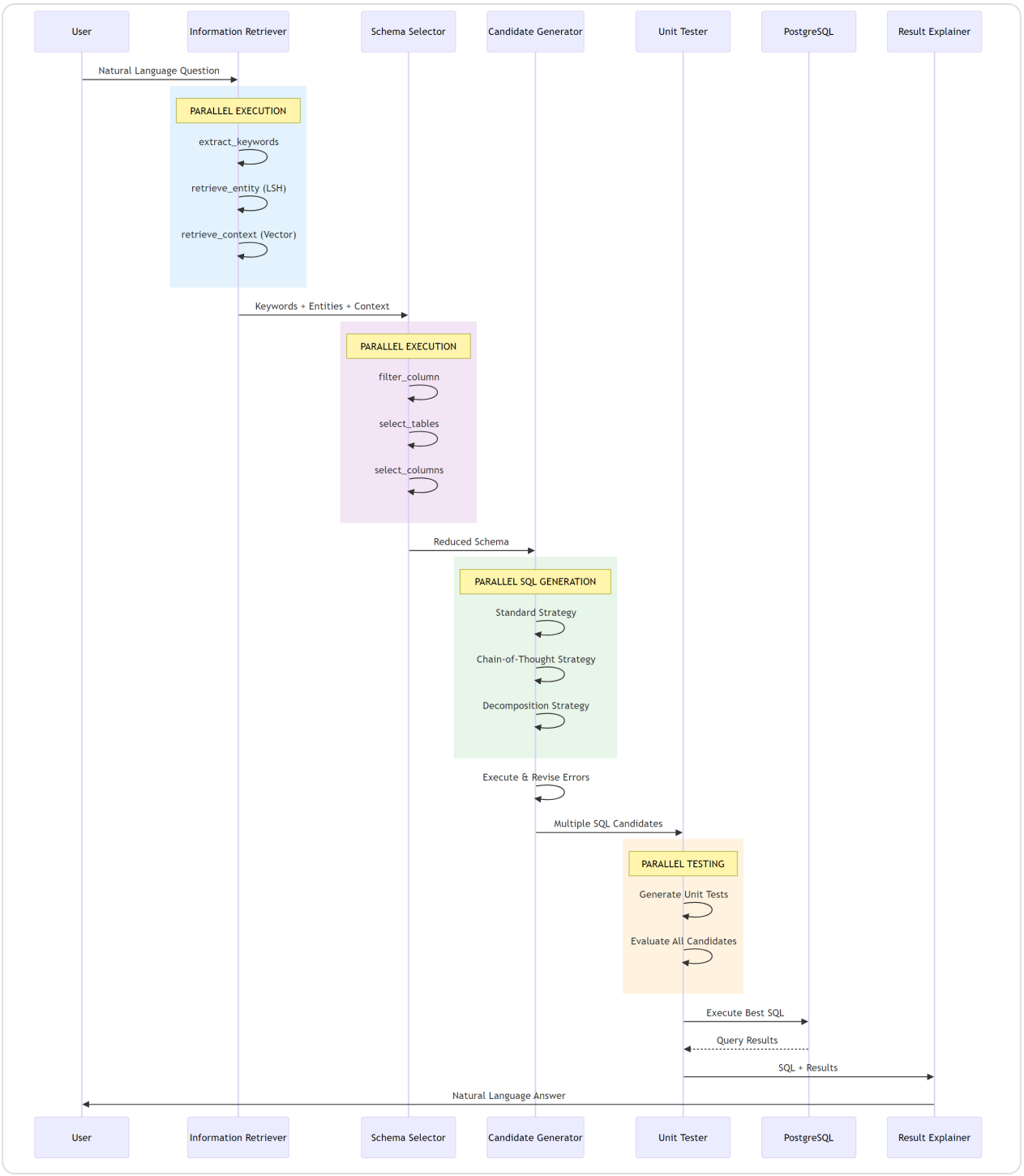
Table of Contents

1. Architecture Overview

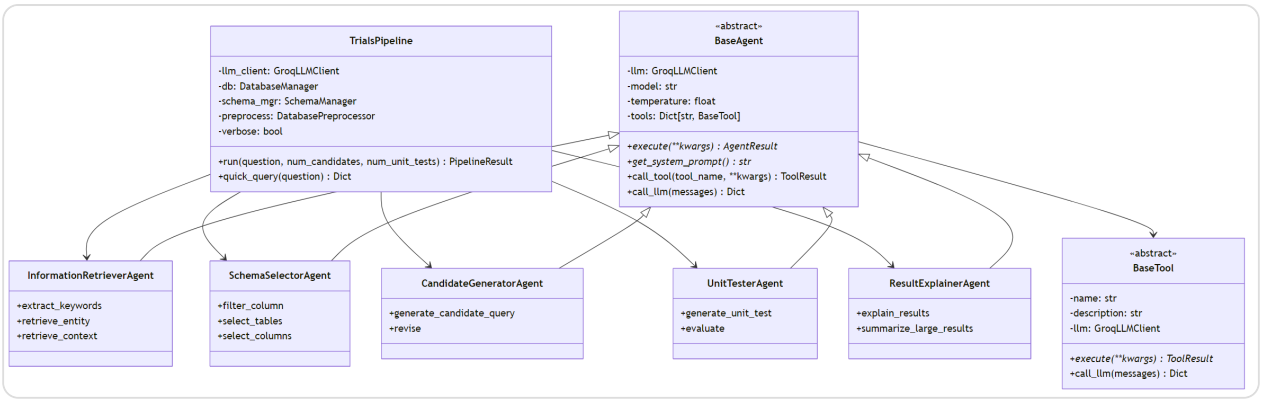
1.1 System Overview Diagram



1.2 Parallel Execution Flow

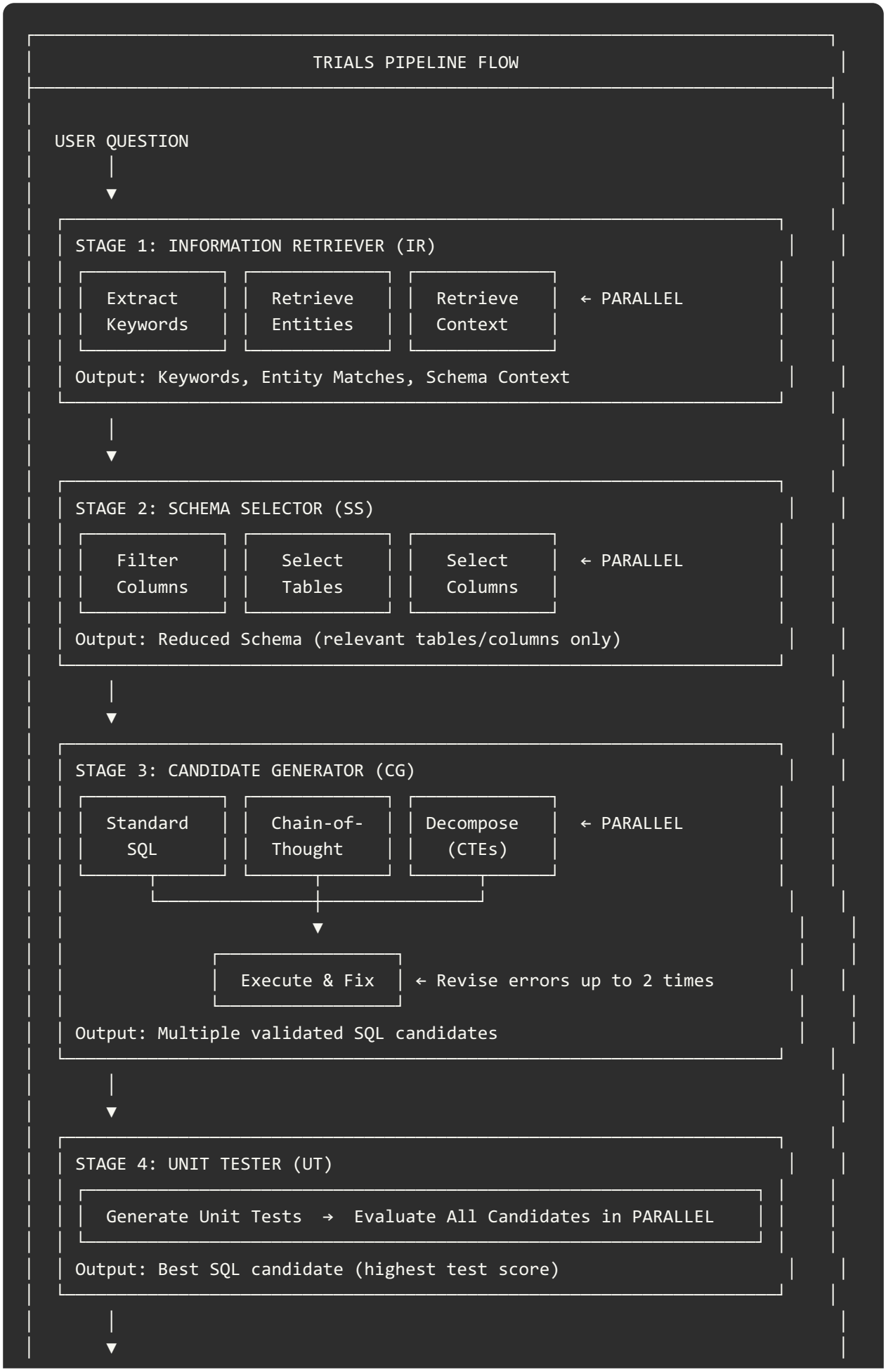


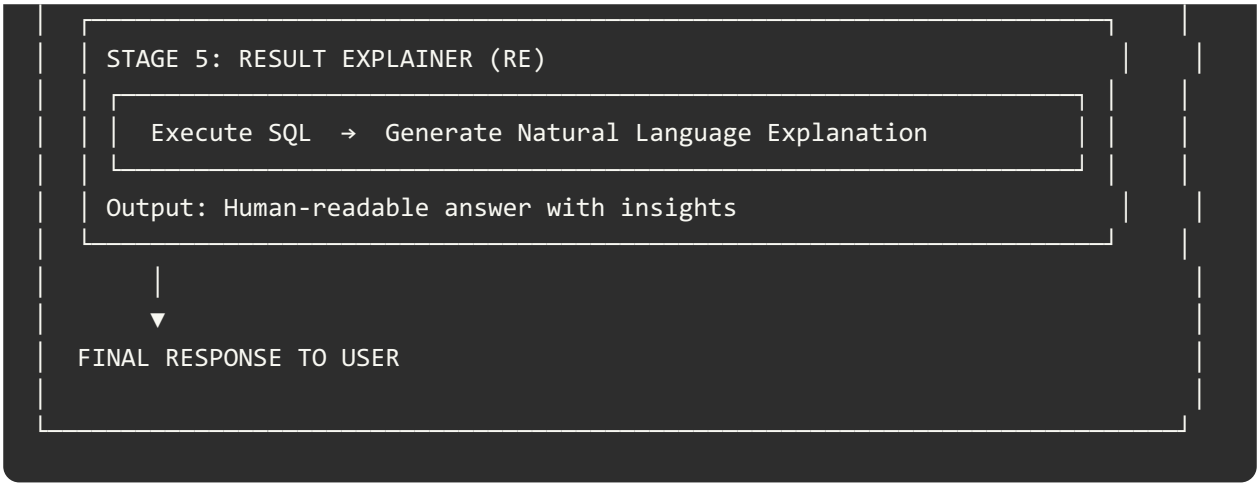
1.3 Component Architecture



2. Pipeline Flow

The TRIALS pipeline processes queries through 5 sequential stages:


TRIALS PIPELINE FLOW
USER QUESTION
STAGE 1: INFORMATION RETRIEVER (IR)
Extract Keywords
Retrieve Entities
Retrieve Context
← PARALLEL
Output: Keywords, Entity Matches, Schema Context
STAGE 2: SCHEMA SELECTOR (SS)
Filter Columns
Select Tables
Select Columns
← PARALLEL
Output: Reduced Schema (relevant tables/columns only)
STAGE 3: CANDIDATE GENERATOR (CG)
Standard SQL
Chain-of-Thought
Decompose (CTEs)
← PARALLEL
Execute & Fix
← Revise errors up to 2 times
Output: Multiple validated SQL candidates
STAGE 4: UNIT TESTER (UT)
Generate Unit Tests → Evaluate All Candidates in PARALLEL
Output: Best SQL candidate (highest test score)



3. Agent Documentation

3.1 Agent 1: Information Retriever (IR)

Purpose: Gathers all relevant information from the database and question before SQL generation.

Location: `agents/information_retriever.py`

Tools

Tool	Description	Input	Output	Uses LLM
`extract_keywords`	Extracts primary keywords and key phrases from the natural language question using few-shot LLM prompting	Question string	Keywords, entities, clinical terms, filters	Yes
`retrieve_entity`	Searches for similar values in the database using LSH (Locality Sensitive Hashing) + edit distance	Keywords list	Matched entities with table/column info	No
`retrieve_context`	Gets relevant schema descriptions from vector database using semantic similarity	Question string	Context items, relevant tables	No

Keyword Extraction Output Format

```
{
    "keywords": ["patients", "study", "count"],
    "entities": ["Study 5", "5"],
    "clinical_terms": ["patients", "enrollment"],
    "filters": ["active", "completed"]
}
```

Entity Retrieval Algorithm

1. For each keyword, query the LSH index
2. Get candidate matches from hash tables
3. Compute edit distance similarity for candidates
4. Return top-k matches with table/column metadata (threshold: 0.5)

3.2 Agent 2: Schema Selector (SS)

Purpose: Reduces the database schema to only relevant tables and columns, minimizing context sent to the SQL generator.

Location: `agents/schema_selector.py`

Tools

Tool	Description	Input	Output	Uses LLM
`filter_column`	Determines if columns are relevant to the query. Processes in batches for efficiency	Columns list, question	Relevant columns	Yes (fast)
`select_tables`	Selects necessary tables from the schema using LLM reasoning	Schema, question, keywords	Selected tables with roles	Yes
`select_columns`	Narrows down to essential columns per table	Table, columns, question	Essential columns with usage	Yes (fast)

Table Roles

Role	Description
primary	Main table containing the data being queried
join	Table needed for JOINs to connect data
filter	Table used only for WHERE conditions

3.3 Agent 3: Candidate Generator (CG)

Purpose: Generates multiple SQL query candidates using different strategies and revises faulty ones.

Location: `agents/candidate_generator.py`

SQL Generation Strategies

Strategy 1: Standard

Property	Value
Temperature	0.1 (low randomness)
Best for	Simple to moderate complexity queries
Approach	Direct SQL generation with multi-step reasoning

Strategy 2: Chain-of-Thought (CoT)

Property	Value
Temperature	0.2 (slightly more creative)
Best for	Complex queries requiring logical reasoning
Approach	Explicit step-by-step reasoning before SQL

Steps:
1. Understand the question
2. Identify tables
3. Identify columns
4. Plan JOINs
5. Plan filters
6. Plan aggregations
7. Write the query

Strategy 3: Decomposition

Property	Value
Temperature	0.15
Best for	Multi-part analytical queries
Approach	Breaks complex queries into CTEs

Output Format:

```
WITH
  step1 AS (SELECT ...),
  step2 AS (SELECT ... FROM step1 ...),
  step3 AS (SELECT ... FROM step2 ...)
SELECT ... FROM step3 ...;
```

Revise Tool - Common Issues Fixed

1. **Column not found** - Check schema for correct column names
2. **Table not found** - Check schema for correct table names
3. **Syntax error** - Fix SQL syntax (commas, brackets, keywords)
4. **Type mismatch** - Ensure comparing same types, use CAST if needed
5. **Ambiguous column** - Add table alias prefix
6. **GROUP BY error** - Include all non-aggregated SELECT columns
7. **Empty result** - Check WHERE conditions

3.4 Agent 4: Unit Tester (UT)

Purpose: Selects the best SQL candidate by generating and running unit tests to differentiate between candidates.

Location: `agents/unit_tester.py`

Unit Test Types

Type	What It Checks
`columns`	Column names, data types
`aggregation`	COUNT, SUM, AVG correctness
`filter`	Filter conditions, operators
`join`	Join keys, table relationships
`result_type`	Row count expectations, formats

Selection Methods

Method	When Used	Description
`single_valid`	Only 1 valid candidate	Direct selection, no testing needed
`unit_test_scoring`	Multiple valid candidates	Score based on tests passed
`fallback_first_valid`	Test generation fails	Use first valid candidate
`best_effort`	No valid candidates	Use first candidate anyway

3.5 Agent 5: Result Explainer (RE)

Purpose: Converts SQL query results into natural language explanations that directly answer the user's question.

Location: `agents/result_explainer.py`

Explanation Guidelines

1. Start with a direct answer to the user's question
2. Provide key insights from the data
3. Mention notable patterns, trends, or outliers
4. Use specific numbers and values from the results
5. If results are sampled, mention there are more rows
6. Keep explanations concise but informative
7. Format numbers nicely (percentages, counts)
8. If result is empty, explain what that means

---

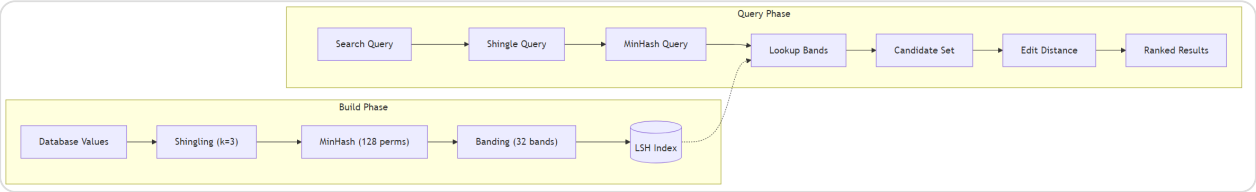
4. Preprocessing System

The preprocessing module handles indexing for fast retrieval operations.

Location: `preprocessing/indexer.py`

4.1 LSH (Locality Sensitive Hashing) Index

Used for fast approximate string matching when retrieving entities from the database.

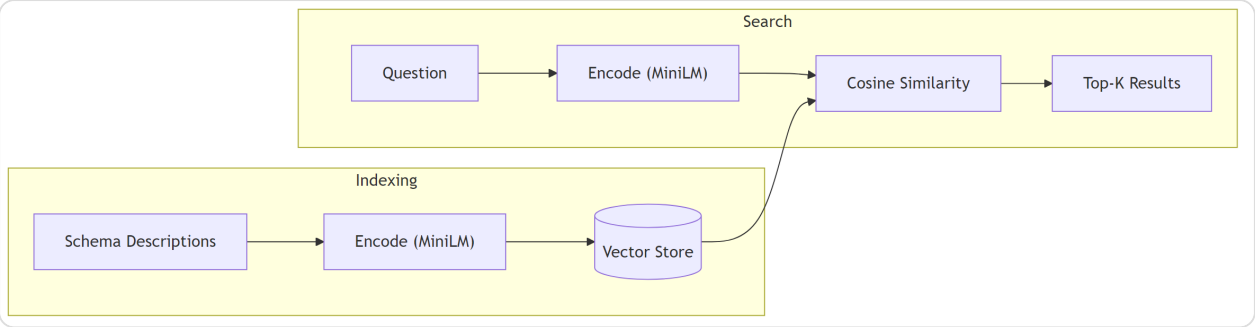


LSH Configuration

Parameter	Value	Description
`num_perm`	128	Number of MinHash permutations
`threshold`	0.5	Minimum similarity threshold
`num_bands`	32	Number of bands for LSH
`rows_per_band`	4	Rows per band (128/32)
`shingle_size`	3	Character n-gram size

4.2 Vector Store

Used for semantic similarity search over schema descriptions.



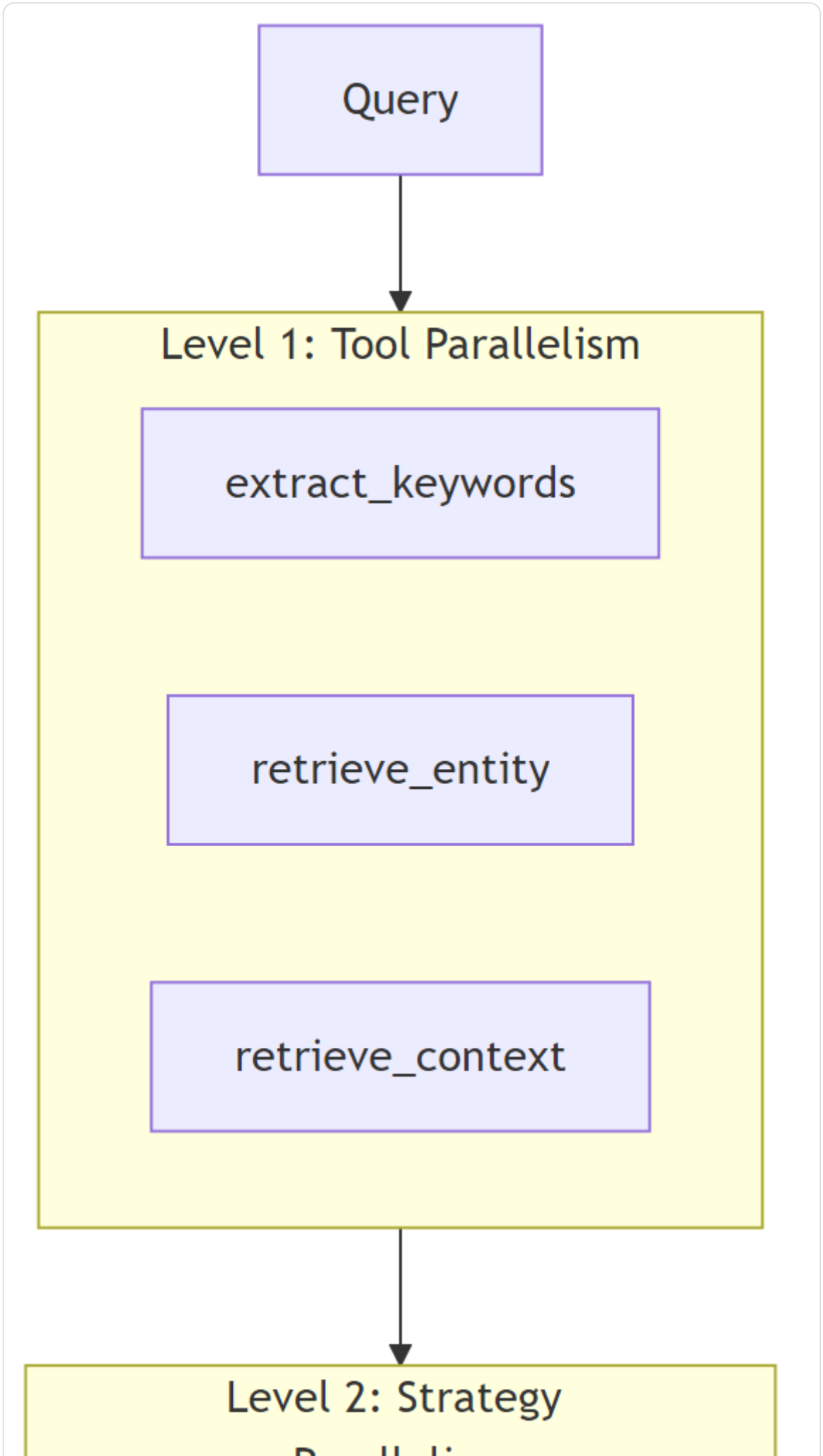
Vector Store Configuration

Parameter	Value	Description
Model	`all-MiniLM-L6-v2`	Sentence transformer model
Embedding Dim	384	Output embedding dimensions
Similarity	Cosine	Similarity metric

5. Parallel Execution

TRIALS maximizes efficiency through parallel execution at multiple levels.

5.1 Parallelism Levels


Query
Level 1: Tool Parallelism
extract_keywords
retrieve_entity
retrieve_context
Level 2: Strategy

5.2 Performance Impact

Stage	Sequential Time	Parallel Time	Speedup
IR Agent	~3s	~1.2s	**2.5x**
CG Agent	~9s	~3.5s	**2.6x**
UT Agent	~5s	~2s	**2.5x**
Total	**~17s**	**~7s**	**2.4x**

---

6. Configuration

6.1 Model Configuration

```
MODELS = {
    "schema_selector": "llama-3.1-8b-instant",   # Fast model for schema selection
    "sql_generator": "llama-3.3-70b-versatile",  # Powerful model for SQL generatio
    "sql_refiner": "llama-3.1-8b-instant",       # Fast model for error fixing
    "evaluator": "llama-3.3-70b-versatile"       # For evaluation tasks
}
```

6.2 Token Limits

Setting	Value	Description
`max_schema_tokens`	4000	Max tokens for schema context
`max_examples_tokens`	1500	Max tokens for few-shot examples
`max_query_tokens`	500	Max tokens for generated query
`total_context_limit`	8000	Total context window limit

6.3 Agent Configuration

Setting	Value	Description
`max_retries`	3	Max revision attempts
`temperature`	0.1	Default temperature
`top_candidates`	3	Number of SQL candidates to generate

---

7. Data Structures

7.1 PipelineResult

```
@dataclass
class PipelineResult:
    success: bool                    # Whether pipeline succeeded
    sql: str                         # Generated SQL query
    question: str                    # Original question
    execution_result: Dict           # Query execution results
    explanation: str                 # Natural language explanation

    # Agent results for transparency
    ir_result: AgentResult           # Information Retriever output
    ss_result: AgentResult           # Schema Selector output
    cg_result: AgentResult           # Candidate Generator output
    ut_result: AgentResult           # Unit Tester output
    re_result: AgentResult           # Result Explainer output

    # Metrics
    total_tokens: int                # Total LLM tokens used
    total_time: float                # Total execution time
    error: str                       # Error message if failed
```

7.2 AgentResult

```
@dataclass
class AgentResult:
    success: bool                    # Whether agent succeeded
    data: Dict                       # Agent output data
    reasoning: str                   # Agent's reasoning/explanation
    tokens_used: int                 # Tokens used by this agent
    execution_time: float            # Time taken
    tool_calls: List[ToolResult]     # Individual tool results
    error: str                       # Error if failed
```

7.3 SQL Candidate

```
{
    "sql": str,                      # The SQL query
    "strategy": str,                 # Generation strategy used
    "is_valid": bool,                # Whether query executes successfully
    "error": str,                    # Error message if invalid
    "result_preview": {              # Preview of execution results
        "columns": List[str],
        "row_count": int,
        "sample_rows": List[Dict]
    },
    "was_revised": bool              # Whether query was revised
}
```

---

8. Benchmarking

8.1 TRIALS-BENCH Overview

TRIALS-BENCH is a comprehensive evaluation framework for testing Text-to-SQL systems on clinical trial data.

- **25 curated test cases** across 3 difficulty levels
- **8 clinical trial tables** covering study metrics, safety data, coding records
- **Multiple query categories**: count, aggregation, filtering, joins, complex analytics

8.2 Difficulty Levels

Level	Count	Description
Easy	10	Single-table queries with basic counts and filters
Medium	10	Group-by aggregations, distinct counts, filtered max
Hard	5	Multi-table joins, percentages, complex analytics

8.3 Query Categories

Category	Description
`count`	Basic record counting
`aggregation`	SUM, MAX, AVG operations
`count_filter`	Counting with WHERE conditions
`count_distinct`	Distinct value counting
`group_by_max`	Finding top values per group
`percentage`	Ratio calculations
`multi_table_sum`	Cross-table aggregations
`top_n`	Ranking queries

9. Technical Specifications

9.1 Clinical Trial Data Categories

Category	Description	Example Tables
visit	Patient visit tracking and projections	`study_X_visit_projection`
query	Data queries and EDRR metrics	`study_X_edrr`, `study_X_query_status`
safety	eSAE and safety data	`study_X_esae`, `study_X_safety_dashboard`
coding	Medical coding (MedDRA, WHODD)	`study_X_meddra`, `study_X_whodd`
lab	Laboratory data and reconciliation	`study_X_lab_recon`
edc_metrics	EDC performance metrics	`study_X_edc_metrics`
forms	Form status (frozen, locked, signed)	`study_X_forms`
pages	Missing pages reports	`study_X_missing_pages`

9.2 Project Structure

```
TRIALS/
├── agents/                     # Agent implementations
│   ├── base_agent.py          # Base classes for agents and tools
│   ├── information_retriever.py  # IR Agent
│   ├── schema_selector.py     # SS Agent
│   ├── candidate_generator.py # CG Agent
│   ├── unit_tester.py         # UT Agent
│   └── result_explainer.py    # RE Agent
├── config/
│   └── settings.py            # Configuration settings
├── database/
│   ├── connection.py          # PostgreSQL connection manager
│   ├── data_loader.py         # Excel to PostgreSQL loader
│   └── schema_manager.py      # Schema extraction and caching
├── preprocessing/
│   └── indexer.py             # LSH and Vector DB indices
├── pipeline/
│   └── orchestrator.py        # Pipeline coordinator
├── trials_bench/              # TRIALS-BENCH evaluation framework
│   ├── run_evaluation.py      # Evaluation runner
│   └── new_testbench.json     # Test cases (25 questions)
├── api/
│   └── server.py              # REST API server
├── cli/
│   └── main.py                # Command-line interface
├── utils/
│   ├── llm_client.py          # Groq API client
│   └── token_utils.py         # Token counting utilities
├── trials_sql.py              # Main entry point
└── requirements.txt
```

9.3 Token Optimization Strategies

1. **Keyword Extraction**: Only relevant terms used for retrieval
2. **Schema Filtering**: Only necessary tables/columns included
3. **Progressive Detail**: Compact to detailed schema as needed
4. **Batched Filtering**: Column filtering in batches of 20
5. **Result Sampling**: Large results sampled before explanation
6. **Caching**: Agent results cached to avoid recomputation

Appendix A: Example Execution Trace

```
User: "How many patients are in Study 5?"

═══════════════════════════════════════════════════════════════
STAGE 1: Information Retriever
═══════════════════════════════════════════════════════════════
  ├─ extract_keywords: {"keywords": ["patients", "study"], "entities": ["Study 5"]}
  ├─ retrieve_entity: Found "Study 5" in study_5_subject_level_metric
  └─ retrieve_context: Relevant tables: study_5_subject_level_metric (0.87)

═══════════════════════════════════════════════════════════════
STAGE 2: Schema Selector
═══════════════════════════════════════════════════════════════
  ├─ select_tables: study_5_subject_level_metric (primary)
  └─ select_columns: subject_id, site_id, country

═══════════════════════════════════════════════════════════════
STAGE 3: Candidate Generator (PARALLEL)
═══════════════════════════════════════════════════════════════
  ├─ Candidate 1 (standard): SELECT COUNT(DISTINCT subject_id)... ✓ Valid
  ├─ Candidate 2 (cot): SELECT COUNT(*) FROM... ✓ Valid
  └─ Candidate 3 (decomposition): WITH patients AS... ✓ Valid

═══════════════════════════════════════════════════════════════
STAGE 4: Unit Tester (PARALLEL)
═══════════════════════════════════════════════════════════════
  ├─ Generated 5 unit tests
  ├─ Candidate 1: 5/5 tests passed ★
  ├─ Candidate 2: 3/5 tests passed
  └─ Candidate 3: 4/5 tests passed

  Selected: Candidate 1

═══════════════════════════════════════════════════════════════
STAGE 5: Result Explainer
═══════════════════════════════════════════════════════════════
  SQL: SELECT COUNT(DISTINCT subject_id) FROM study_5_subject_level_metric
  Result: 150

  Explanation: "Based on the query results, there are **150 patients**
  enrolled in Study 5. The data shows patients distributed across
  12 sites in 5 different countries."

═══════════════════════════════════════════════════════════════
METRICS
═══════════════════════════════════════════════════════════════
  Total Time: 6.8s
  Total Tokens: 4,250
  IR: 1.2s, 850 tokens
  SS: 0.9s, 620 tokens
  CG: 3.1s, 1,890 tokens
```

```
  UT: 1.4s, 740 tokens
  RE: 0.2s, 150 tokens
```

Appendix B: Example Queries

```
# Patient/Site Queries
"How many patients are in Study 5?"
"Which sites have the most missing visits?"
"Show patient enrollment by region"

# Data Quality Queries
"How many open queries are there?"
"Which sites have non-conformant data?"
"Show query resolution rates by site"

# Safety Queries
"List eSAE events by study"
"Which patients have adverse events?"

# Aggregation Queries
"Show missing visit percentages by site"
"Calculate clean patient rates"
```

Document generated for Novartis Clinical Intelligence Platform
TRIALS Pipeline v1.0.0