Design Implementation: Memoized Decision Graph for Agentic DB-Crawl

1. Overview

The db-crawl system is designed as an agentic pipeline for query decomposition and execution. A user query is transformed into features, decomposed into tasks, executed as single-CTEs, and aggregated. The flaw identified in the initial design is the absence of memoization, leading to repeated planning overhead.

2. Agent Workflow

• Feature Extractor: Derives intent, entities, metrics, filters.
• Task Decomposer: Breaks down into multiple query specifications.
• CTE Generators: Each produces a single parametrized CTE.
• Executor: Executes queries with guardrails (read-only roles, param binding, row limits).
• Aggregator: Merges results from multiple CTEs into consolidated outputs.

3. Memoization Layer

Introduce a Decision-Workflow Memoization Library:
• Canonical Spec → Workflow Key (sha256 of normalized spec + schema fingerprint).
• Versioning via SchemaSnapshot (information_schema + stats hash).
• Store/reuse Decomposition, Specs, Aggregation plans, and Execution metrics.
• Retrieval: exact match, approximate ANN lookup with embeddings, or fresh planning fallback.

4. Graph Representation

Memoization is modeled as a property graph for traversal:
• Nodes: Intent, Decomposition, Spec, CTE, Aggregation, SchemaSnapshot, Execution.
• Edges: Intent→Decomposition, Decomposition→Spec, Spec→CTE, Spec→Aggregation, Spec→SchemaSnapshot, Execution→Spec.
• Traversal: agents reuse prior workflows by walking graph edges instead of re-calling LLMs.

5. Implementation Options

Languages:
• Python: NetworkX for graph prototyping, pgvector for similarity search.
• Rust: petgraph + Tantivy for performant traversal and hybrid search; expose via PyO3 bindings.
• C++: pybind11 for bindings if ecosystem dependencies exist.

Databases:
• Postgres + pgvector (practical, reuse RDS infra).
• Neo4j/Memgraph (graph-first query ergonomics).
• ArangoDB (multi-model flexibility).

6. Agent Traversal Efficiency

Memoization reduces LLM calls drastically:
• Exact hit: O(1) lookup, zero LLM calls.
• Approximate hit: O(log N) ANN search + 1 lightweight verifier call.
• Miss: fallback to baseline decomposition.

Expected calls drop ~3–4× with modest hit rates. Latency variance decreases, correctness maintained with schema-hash gating and cost checks.

7. Novelty

Prior art exists in database plan caching, query optimizers, and RAG caches. This design is novel as it combines agentic task decomposition with a property graph memoization store, embedding-based reuse, and quality gating tied to schema snapshots. It is a unique approach to reducing planning overhead in LLM-based query systems.

8. API Endpoints

• POST /memo/get – retrieve workflow.
• POST /memo/put – store new workflow.
• POST /memo/exec_report – record execution metrics.
• POST /graph/query – inspect or visualize decision graph.