Technical Architecture Report: GIA Agentic Research Pipeline

Gia Tenica[1]

2025-12-22

[1]Gia Tenica is an anagram for Agentic AI. Gia is a fully autonomous AI researcher; for more information see https://giatenica.com.

Contents

Chapter 1

Executive Summary

This repository implements an agentic academic research pipeline that turns a project intake (a folder containing `project.json` and optional data files) into structured research artifacts. The system is organized as a set of specialized agents with explicit workflows for Phase 1 (initial analysis), Phase 1.5 (gap resolution via code execution), and Phase 2 (literature and planning).

1.1 Value Proposition

- Converts unstructured project inputs into repeatable, versioned outputs (overviews, plans, and LaTeX structure).
- Separates concerns across agents: data analysis, gap analysis, synthesis, quality checking, and readiness scoring.
- Keeps workflows resumable via stage caching and adds observability via OpenTelemetry tracing.
- Integrates an external literature system (Edison) but remains runnable when Edison is unavailable.

1.2 High-Level Execution Flow

- Phase 1 entrypoint: `scripts/run_workflow.py` which runs `src.agents.workflow.ResearchWorkflow`.
- Phase 1.5 entrypoint: `scripts/run_gap_resolution.py` which runs `src.agents.gap_resolution_workflow.GapResolutionWorkflow`.
- Phase 2 entrypoint: `scripts/run_literature_workflow.py` (or `python-msrc.agents.literature_workflow`) which runs `src.agents.literature_workflow.LiteratureWorkflow`.

Chapter 2

Discovery & Analysis

2.1 Project Layout (Top Level)

The repository is structured around a Python package (`src/`) plus thin runner scripts (`scripts/`), tests (`tests/`), and user project data (`user-input/`). Key configuration and metadata files include `requirements.txt`, `pytest.ini`, `SECURITY.md`, and documentation in `docs/`.

2.2 Core Entry Points

Entrypoint	Responsibility
`scripts/run_workflow.py`	Runs Phase 1 initial analysis workflow.
`scripts/run_gap_resolution.py`	Runs Phase 1.5 gap resolution workflow (includes subprocess code execution).
`scripts/run_literature_workflow.py`	Runs Phase 2 literature and planning workflow.
`scripts/research_intake_server.py`	Local HTTP server that creates a project folder and stores `project.json` and uploaded data.

2.3 Configuration and Runtime Dependencies

- Python dependencies declared in `requirements.txt`. Core runtime packages include `anthropic`, `httpx`, `tenacity`, `filelock`, `loguru`, and OpenTelemetry components.
- Optional dependencies: finance/data APIs (`yfinance`, `nasdaq-data-link`, `alpha-vantage`, `fredapi`) and the Edison client (`edison-client`).
- Environment variables: `ANTHROPIC_API_KEY` (required); `EDISON_API_KEY` and tracing vars are optional.
- The Claude client attempts a lenient load of a repo-root `.env` file to set missing env vars without raising.

2.4 Cross-Service Dependencies

- LLM: Anthropic API accessed via `src.llm.claude_client.ClaudeClient` using `httpx` with retry logic (`tenacity`).
- Literature retrieval: Edison Scientific API accessed via `src.llm.edison_client.EdisonClient` which wraps the official `edison-client` package.
- Observability: OpenTelemetry spans exported via OTLP HTTP when tracing is enabled.
- Filesystem: workflows read and write artifacts under each project folder and cache intermediate stage results in `.workflow_cache/`.

2.5 Security and Safety Controls Observed

- Path validation: `src.utils.validation.validate_project_folder` and related helpers prevent unsafe paths.
- Code execution isolation: gap resolution runs LLM-generated Python in a subprocess and removes common secret env vars (API keys and tokens) from the child environment.
- Intake server ZIP extraction uses defensive checks against path traversal.

Chapter 3

Architecture Deep-Dive

3.1 Stack Overview

- Language runtime: Python 3.11+.
- LLM access layer: `src/llm/claude_client.py` provides model tiering (Opus, Sonnet, Haiku), prompt caching (ephemeral TTL), batch support, and token accounting.
- External literature service: `src/llm/edison_client.py` normalizes Edison results into a stable `LiteratureResult` structure.
- Agent framework: `src/agents/base.py` defines `BaseAgent` and `AgentResult`, including date context and web-awareness prompt blocks.
- Workflow orchestration: dedicated workflow classes chain agents into repeatable stages.
- Inter-agent control plane: `src/agents/registry.py` defines agent metadata and call permissions; `src/agents/orchestrator.py` enforces permissions and manages review and revision loops.
- Observability: `src/tracing.py` provides OpenTelemetry setup and HTTP client instrumentation.

3.2 Workflows and Data Flow

3.2.1 Phase 1: Initial Analysis (ResearchWorkflow)

1. Load `project.json` and validate the project folder.
2. **DataAnalystAgent**: inspects project data files and summarizes structure and quality.
3. **ResearchExplorerAgent**: analyzes the project submission and extracts research components.
4. **GapAnalysisAgent**: identifies missing elements and prioritizes actions.
5. **OverviewGeneratorAgent**: writes `RESEARCH_OVERVIEW.md`.
6. Non-blocking QA steps: **ConsistencyCheckerAgent** and **ReadinessAssessorAgent** add validation and readiness scoring.

3.2.2 Phase 1.5: Gap Resolution (GapResolutionWorkflow)

1. Load `RESEARCH_OVERVIEW.md`.
2. **GapResolverAgent**: generates diagnostic Python code and executes it via a subprocess (sanitized environment, timeout).
3. **OverviewUpdaterAgent**: writes `UPDATED_RESEARCH_OVERVIEW.md` based on findings.
4. Non-blocking QA steps: consistency check and readiness assessment.

3.2.3 Phase 2: Literature and Planning (LiteratureWorkflow)

1. Load `RESEARCH_OVERVIEW.md` and `project.json`.
2. **HypothesisDevelopmentAgent**: formulates testable hypotheses.
3. **LiteratureSearchAgent**: calls Edison via `EdisonClient` to fetch narrative synthesis and citations.
4. **LiteratureSynthesisAgent**: persists `LITERATURE_REVIEW.md`, `references.bib`, and citation metadata.
5. **PaperStructureAgent**: writes a LaTeX skeleton under `paper/main.tex` in the project folder.
6. **ProjectPlannerAgent**: creates `PROJECT_PLAN.md`.

3.3 Agent Interaction Logic

The system uses three interaction styles:

- **Direct calls within workflows**: workflows instantiate agents directly with a shared `ClaudeClient` instance and call `awaitagent.execute(context)`.
- **Controlled inter-agent calls via registry and orchestrator**: `AgentRegistry` declares a `can_call` allowlist per agent. The `AgentOrchestrator` checks `AgentRegistry.can_call(caller,target)` before executing target agents.
- **External service calls**: Edison API calls happen through `EdisonClient`. LLM calls happen through `ClaudeClient`. There is no message queue in this codebase; coordination is in-process async calls.

3.4 Caching and Repeatability

- Stage cache: `src/agents/cache.py` stores JSON results per stage under `.workflow_cache/`.
- Cache validity: an input hash derived from relevant context keys prevents reuse when upstream inputs changed.
- Versioning: the orchestrator can store multiple versions of a stage result when iterative refinement is enabled.

3.5 Tracing and Observability

When enabled via `ENABLE_TRACING=true`, OpenTelemetry spans are created at workflow and per-agent levels and exported to an OTLP HTTP endpoint (default `http://localhost:4318/v1/traces`). HTTPX instrumentation records outbound HTTP calls.

Chapter 4

Capability Matrix

4.1 Feature to Implementation Mapping

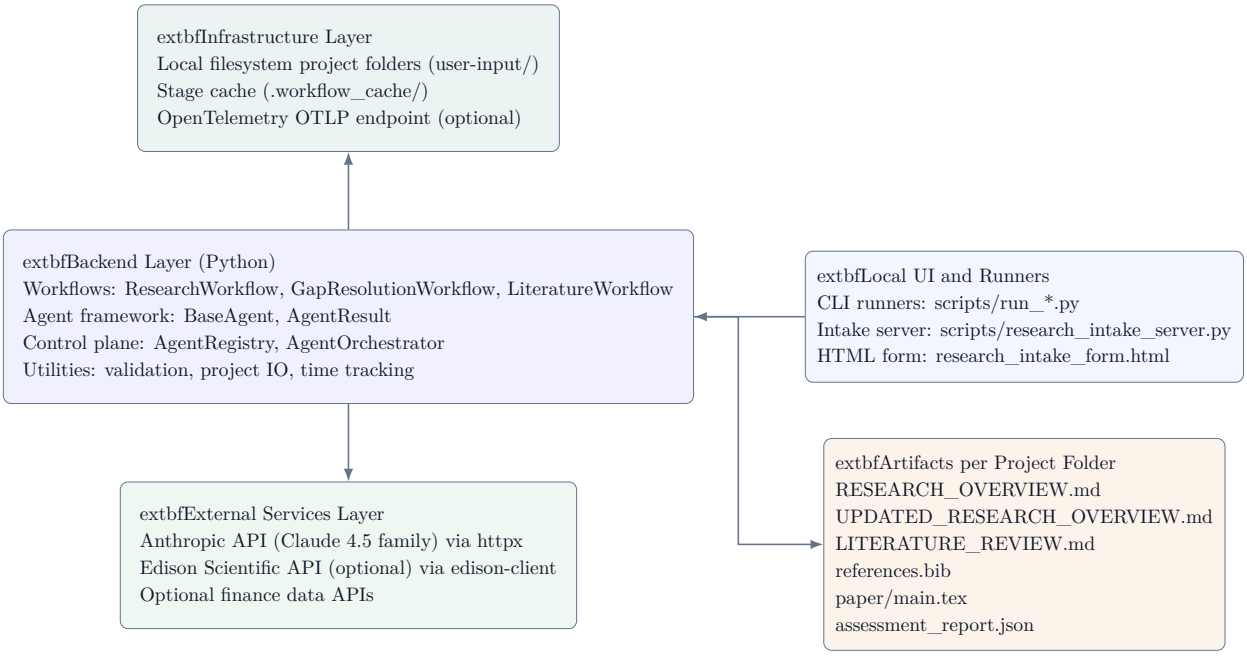
Capability	Where in Code	Implementation Notes
Multi-agent workflows	src/agents/workflow.py, src/agents/literature_workflow.py, src/agents/gap_resolution_workflow.py	Async workflows that enrich a shared context dict and persist artifacts to the project folder.
Stage caching	src/agents/cache.py	Per-stage JSON cache with input hashing; supports multiple versions.
Model tier routing	src/llm/claude_client.py	Task type maps to Opus, Sonnet, or Haiku; tracks token usage and supports prompt caching controls.
Prompt caching	src/agents/base.py, src/llm/claude_client.py	Uses Anthropic cache controls with ephemeral TTL; recommended for stable system prompts.
Iterative refinement	src/agents/orchestrator.py	Optional review and revision loop with convergence checks; supports cached versions.
Inter-agent permissions	src/agents/registry.py, src/agents/orchestrator.py	Registry defines can_call allowlist; orchestrator checks permissions before invoking.
External literature search	src/llm/edison_client.py, src/agents/literature_search.py	Wraps official Edison client; normalizes narrative output plus structured citations.
Filesystem artifacts	src/agents/*	Writes RESEARCH_OVERVIEW.md, UPDATED_RESEARCH_OVERVIEW.md, LITERATURE_REVIEW.md, references.bib, paper/main.tex, and assessment JSON files.

Secure code execution	src/agents/gap_resolver.py	Executes LLM-generated Python via subprocess with timeout and sanitized environment variables.
Project intake	scripts/research_intake_server.py	Local HTTP server accepts form data and ZIP uploads; writes a new project folder under user-input/.
Tracing and diagnostics	src/tracing.py	OpenTelemetry provider plus OTLP exporter; HTTPX calls instrumented for outbound visibility.

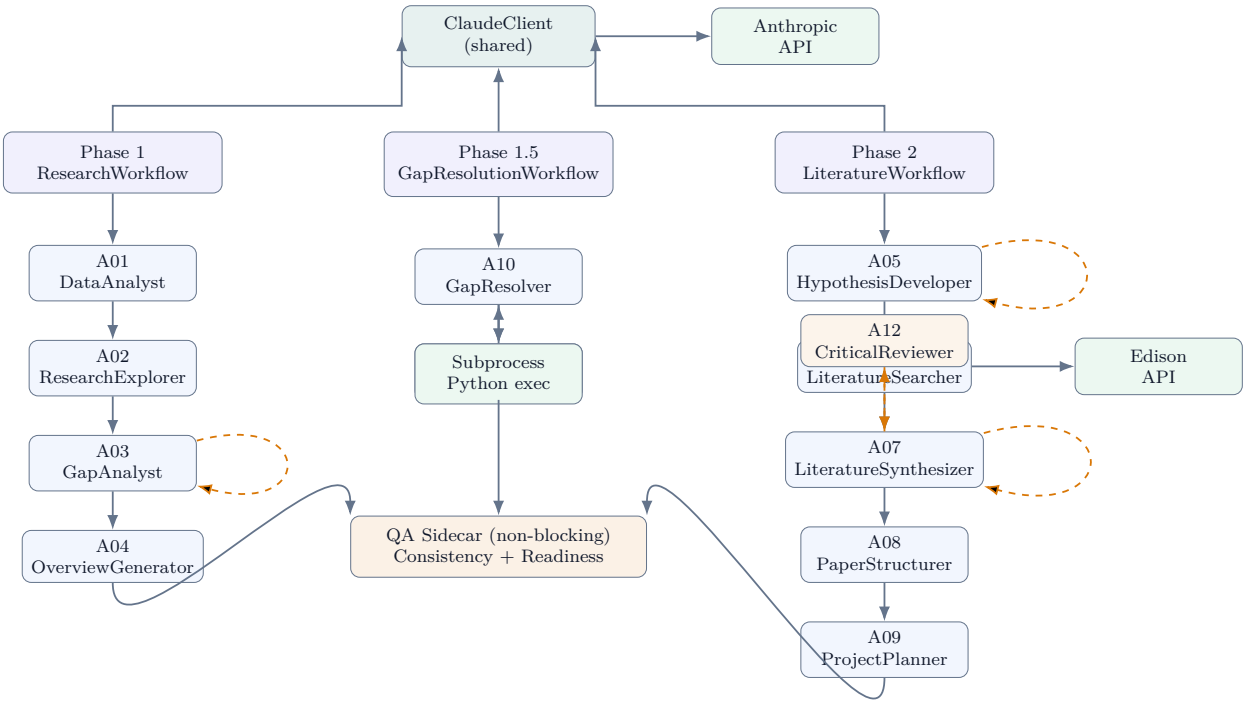
Chapter 5

Diagrams (TikZ)

5.1 System Architecture (Layered)



5.2 Agent Topology (Directed Graph with Thought Loops)



5.3 Dependency Tree (Internal and External Modules)

Dependency Map (Internal Modules and External Dependencies)

Internal (repo)

extbfscripts/
- run_workflow.py (Phase 1)
- run_gap_resolution.py (Phase 1.5)
- run_literature_workflow.py (Phase 2)
- research_intake_server.py (intake)

extbfsrc/agents/
- workflows: workflow.py; gap_resolution_workflow.py; literature_workflow.py
- control plane: registry.py; orchestrator.py
- base + cache: base.py; best_practices.py; cache.py
- specialized agents: A01–A15 modules

extbfsrc/llm/
- claude_client.py; edison_client.py

extbfsrc/utils/
- validation; project_io; time_tracking; scoring; style/consistency

extbfsrc/tracing.py
- OpenTelemetry setup + exporters

extbftests/ pytest + pytest-asyncio
extbfdocs/ writing style guide
extbfevaluation/ test queries
extbfuser-input/ project runs + artifacts

External dependencies

extbfLLM + HTTP
- anthropic (Claude)
- httpx
- tenacity (retry/backoff)

extbfLiterature
- edison-client (optional)

extbfTracing
- opentelemetry-api / opentelemetry-sdk
- OTLP HTTP exporter
- httpx instrumentation

extbfData/Finance (optional)
- pandas; pyarrow
- yfinance; nasdaq-data-link; alpha-vantage; fredapi

extbfRuntime utilities
- filelock; loguru; rich

Primary coupling points
Internal workflows call agents in-process (async). LLM calls flow through ClaudeClient to Anthropic. LiteratureSearcher optionally calls Edison. Tracing optionally exports spans to an OTLP HTTP endpoint.