The **Self-Improving Research and Coding Agent (RCA) Framework** is a distributed, auditable multi-agent system designed for autonomous code generation, advanced research, quality assurance (CI/CD), and continuous self-improvement. It leverages GraphRAG for contextual knowledge retrieval, with all operations governed by a central Supervisor Agent.

1. System Architecture and Supervisory Control

The RCA operates on a strict, centralized command-and-control model where the **Supervisor Agent** dictates all task execution and communication.

1.1. Core Agent Components and Mandates

Agent Component	Core Functionality	Supervisory Control Mandate
SupervisorAgent	**Central Command & Audit:** Manages concurrent execution of **User Projects (Workflow A)** and **Framework Improvement (Workflow B)**. Routes all A2A messages and enforces centralized logging.	Logs all system activity to framework.log, all failures to error.log, and all framework improvements to improvement.log.
ProviderAgent	**LLM Router:** Provides vendor-agnostic access to all supported LLMs: OpenAI, Gemini, Grok, Ollama, LM Studio, Deepseek, and Qwen.	Handles credential loading from .env and enforces model selection based on the Supervisor's task delegation.
ResearchAgent	**GraphRAG Knowledge Acquisition:** Retrieves data from external hubs (Kaggle, GitHub) and synthesizes context using **Neo4j** and **GraphRAG**.	Writes acquired data and artifacts to workspace directories for ingestion into Neo4j.
SelfImprovingCodingAgent	**Code Execution & Correction:** Generates user code, runs automated CI/CD verification, and initiates self-correction loops upon compiler/test failure.	Executes CI/CD tools (subprocess.run) and reports success/failure metrics back to the Supervisor for logging and routing.
EnvironmentAgent	**Provisioning & Setup:** Ensures the framework is portable across Docker, Anaconda, and native Python environments on Windows, Mac, and Linux, managing all necessary service dependencies (Neo4j, Ollama).	Executes platform-specific installation scripts and reports environment status via A2A.

1.2. Auditable Logging Structure

The Supervisor Agent logs all critical system events to three distinct files in the improvement_logs/ directory, ensuring comprehensive auditability:

Log File	Logged Events	Purpose
framework.log	General application flow, agent status, A2A message traffic, and component initialization.	Operational debugging and system health monitoring.
error.log	All compilation failures, runtime exceptions, A2A communication errors, and security warnings.	Critical audit trail for failure analysis and security incident response.
improvement.log	Successful CI/CD runs on framework code, documentation changes, new semantic version tags, and confirmed knowledge base updates.	Auditing the efficacy and formal versioning of the self-improvement cycle.

2. Installation and Configuration

The framework is configured using a centralized environment file and supports three main installation methods to ensure cross-platform compatibility.

2.1. Prerequisites

1. **Python:** Python 3.10+
2. **Git:** Required for cloning codebases from GitHub.
3. **Neo4j:** A running Neo4j instance (version 5.11+ recommended) is mandatory for the GraphRAG knowledge base.
4. **Docker:** Required for the containerized installation method.

2.2. Configuration (.env File)

Create a .env file in the root directory to store all sensitive and environment-specific settings:

```
# Neo4j Database Credentials
NEO4J_URI=bolt://localhost:7687
NEO4J_USER=neo4j
NEO4J_PASSWORD=your_neo4j_password

# Cloud LLM API Keys (Access managed by ProviderAgent)
OPENAI_API_KEY=sk-...
GOOGLE_API_KEY=AIza...
XAI_API_KEY=grok_... (For Grok)

# External Data Hub Credentials (Access managed by ResearchAgent)
GITHUB_TOKEN=ghp_... (Read-scoped PAT for code search)
HUGGINGFACEHUB_TOKEN=hf_...
KAGGLE_USERNAME=your_username
KAGGLE_KEY=your_api_key
```

2.3. Installation Methods (Managed by Environment Agent)

The EnvironmentAgent facilitates installation on **Windows (via WSL2/Docker Desktop), Mac, and Linux**.

A. Containerized (Docker Compose) - Recommended for Isolation

Docker isolation is critical as the Coding Agent executes unverified code, preventing potential escapes or breaches of the host filesystem.

1. **Install:** Ensure Docker Desktop is running and execute:
   ```
   # Build and start services (Agent, Neo4j, Ollama)
   docker compose up --build -d
   ```
2. **Access:** The agent's shell can be accessed for manual commands:
   ```
   docker exec -it agent-framework /bin/bash
   ```

B. Anaconda (Reproducible Data Science Environment)

1. **Create and Activate Environment:**
   ```
   conda env create -f environment.yml
   conda activate agent-framework
   ```
2. **Start Services:** Manually start the required services (Neo4j Desktop/Server and Ollama if using local LLMs).

C. Native Python Virtual Environment

1. **Setup Environment:**
   ```
   python -m venv.venv
   source.venv/bin/activate
   # Install Python dependencies
   pip install -r requirements.txt
   ```
2. **Start Services:** Manually start Neo4j and Ollama services before running the framework entry script.

3. Usage Examples

Agent interaction relies entirely on the **Supervisor Agent** to delegate tasks using the A2A protocol.

3.1. Example 1: User Code Generation and Research Delegation (Workflow A)

A user requests code for a project. The Coding Agent determines it needs external data and sends an A2A request to the Research Agent, supervised by the central orchestrator.

```
# Assuming the user interacts with the Supervisor CLI/API endpoint:
# 1. Supervisor receives the user's task
user_task = "Implement a Python FastAPI endpoint that retrieves Neo4j
data."

# 2. Supervisor routes the task to the Coding Agent (A2A Task Request)
coding_agent.delegate(task="Generate Fast API code using graph data
model")

# 3. Coding Agent sends a dependency request back through the
Supervisor
#    (A2A Task Request: Find all 'User' and 'Product' nodes schema in
GitHub)
research_task = coding_agent.request_research_context(query="Neo4j
FastAPI tutorial and data model")

# 4. Research Agent performs GraphRAG retrieval using Cypher and
vectors.
retrieved_context = research_agent.execute(research_task)

# 5. Final Code Generation
code_output = coding_agent.generate(
    task=user_task,
    context=retrieved_context,
    output_dir="/user_project_dir/api"
)

print(f"Code saved to: {code_output['user_path']}")
```

3.2. Example 2: Framework Self-Improvement and Auditing (Workflow B)

A new feature is committed to the framework's internal code. CI/CD runs automatically, and the success is logged, increasing the framework's version number.

Sequence	Agent Action	Log File Output
Trigger	Code change detected in framework source.	**framework.log:** INFO: CI/CD Monitor triggered on framework codebase modification.
Verification	Verification Agent runs full test suite (pytest, bandit, flake8).	**framework.log:** INFO: Verification Agent running CI/CD pipeline v2.1.0-beta.
Failure	Test case fails due to a regression.	**error.log:** FATAL: CI/CD Test Failed: framework_parser.py returned exit code 1. Stderr:

Sequence	Agent Action	Log File Output
Correction	Supervisor routes error log to LLM (via ProviderAgent) for fix generation.	**framework.log:** INFO: Correction Loop initiated. Model Grok-code-fast-1 deployed for reasoning.
Integration	Corrected code passes all tests and is merged.	**improvement.log:** SUCCESS: Framework code passed CI/CD. Version incremented from v2.1.0 to v2.1.1 (Patch). New version indexed in Neo4j.

3.3. Example 3: LLM Provider Selection

The Supervisor dynamically selects the LLM provider based on the task type (e.g., highly private local model for proprietary code analysis, or a fast cloud model for research synthesis).

```
# Task requiring complex analysis (Gemini)
supervisor.delegate(
    recipient="ResearchAgent",
    task_type="SYNTHESIZE",
    provider="Gemini",
    model="gemini-1.5-pro"
)

# Task requiring fast, local code generation (Ollama)
supervisor.delegate(
    recipient="SelfImprovingCodingAgent",
    task_type="CODE_GEN",
    provider="Ollama",
    model="codellama:13b-instruct-q4"
)
```