Capstone Project: Mining Regulatory Compliance Assistant – APH-IF

Alexander Ricciardi

Colorado State University Global

CSC480: Capstone Computer Science

Dr. Shaher Daoud

August 3, 2025

Capstone Project: Mining Regulatory Compliance Assistant – APH-IF

The Mining Regulatory Compliance Assistant (MRCA) is an AI-powered web application based on microservices architecture and a novel Retrieval Augmented Generative (RAG) system. MRCA allows users to query Mine Safety and Health Administration (MSHA) regulations (CFR 30) accurately using natural language. The novel Advanced Parallel HybridRAG -Intelligent Fusion (APH-IF), MRCA's novel RAG system, reduces AI hallucinations and improves AI data retrieval accuracy by running in parallel a semantic and a contextual graph traversal searches on a Neo4j Knowledge Graph (KG) database where vector embeddings are stored as attributes on nodes within a KG; and then fuses the search results using Intelligent Fusion (IF), an AI power fusion, to provide a final answer. This final report describes the design, implementation, and evaluation of the MRCA and its novel APH-IF system.

Module 1: The Topic

The MRCA APH-IF project was chosen to demonstrate the skills and knowledge that I acquired by attending the computer science bachelor's program at Colorado State University Global (CSU Global). Moreover, the project was developed to address issues related to AI hallucination and AI information retrieval accuracy in regulatory compliance fields, where accuracy and reliability are essential for ensuring safety and avoiding legal and financial consequences associated with regulatory non-compliance. Such a field is the mining industry, which is regulated by MSHA under Title 30 of the Code of Federal Regulations (CFR). CFR 30 presents an ideal use case to implement APH-IF and test its effectiveness. The proposed MRCA APH-IF application is designed to provide miners, contractors, and safety managers with a quick, reliable, and easy way to query MSHA regulations using natural language, as well as a platform to test the novel APH-IF system.

Module 2: Project Proposal and Project Requirements

The sheer size and complexity of the CFR 30 make it very difficult for mine workers to find quick, reliable, and accurate answers to specific questions about mine regulations. This is especially difficult when on mine sites, where methods such as searching through large printed volumes, digital volumes, or using generic search engines (e.g., Google) are impractical and time-consuming; other methods, such as chatbot powered by Large Language Models (LLMs), often provide unreliable information as they are prone to hallucinations. MRCA APH-IF addresses these issues by providing an AI-powered web application based on the APH-IF system. The APH-IF system grounds the LLM used by MRCA to be grounded to the MSHA regulatory diminishing the risk of AI hallucinations and allowing the LLM to generate more accurate and reliable responses. This makes the MRCA not only a practical, accurate, and reliable tool for querying CFR 30 data, but also a platform for testing the novel APH-IF system.

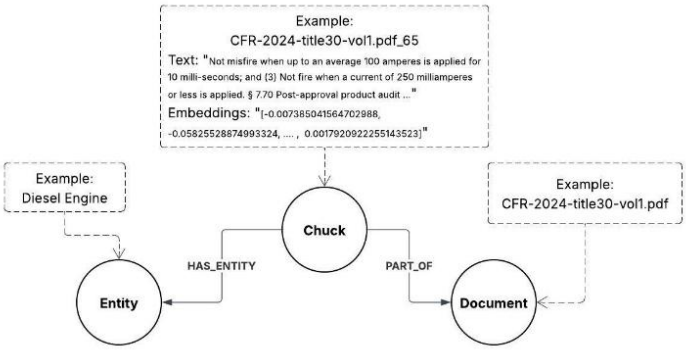
MRCA APH-IF System Overview

The MRCA APH-IF is designed on a modular architecture based on a microservices architecture that isolates major system components into services, ensuring that the codebase is robust, organized, easier to troubleshoot, and maintainable. The APH-IF system combines KG traversal search with semantic vector search using a Neo4j KG database. The database acts as a unified storage system that contains graph nodes, relationships, and vector embeddings. The embeddings and related text are stored in chunks as attributes on graph nodes, see Figure 1. The chunk nodes can be used for either traversal or semantic searches. These searches are done in parallel, and their results are fused using IF (Intelligent Content Fusion), which is performed using Google Gemini 2.5 Pro. The APH-IF system is a type of HybridRAG, which is a RAG system that combines VectorRAG and GraphRAG, and has been shown to have a projected

accuracy of over 92% when using an agentic system that selects how the data is searched, that is, using a traversal (GraphRAG) or semantic (VectorRAG) search, based on the user prompt (Xu et al., 2024). This differs from APH-IF, which performs both searches concurrently and then fuses the results.

Figure 1

Graph Nodes



Note: The diagram illustrates the KG nodes and relationships

Project Stack and Development Timeline

The project's requirements are met by utilizing a specific set of technologies listed below:

- A Neo4j AuraDB stores the KG. Google's Gemini 2.5 Pro in combination with LangChain's `LLMGraphTransformer` are used to generate KB, and Google Gemini embedding-001 model was used to generate a 768-dimensional vector embedding integrated within the chunk nodes.
- OpenAI's GPT4o is used to create cypher queries based on user prompts to perform traversal searches.
- Gemini embedding-001 is used to convert user prompts to vector embedding, and the Neo4j model (`langchain_neo4j.Neo4jVector.as_retriever()`) is used to perform cosine similarity search against stored chunk embeddings, returning the top 5 (k=5) most semantically similar text chunks with a minimal `score_threshold` of 0.7. Then the retrieved chunks are processed

by GPT-4o to generate a response based on their chunks' similarity score, their text content, their related document metadata (source document information), and their related entities (regulatory entities, e.g., Diesel Engine).

- GPT-4o is also used to fuse the search results and provide the final response using templates.
- The backend microservice is A RESTful API built with FastAPI that handles the application logic.
- The frontend microservice handles the User Interface (UI) and chat interface using Streamlit.
- The project is coded in Python 3.12, developed using an iterative Kanban methodology. Version control is managed with Git/GitHub, the development environment is containerized using Docker, and the application is deployed using Render: https://mrca-frontend.onrender.com/

Project Timeline and Major Components

The project was implemented following a strict eight-week timeline based on the Kanban methodology and modular design, meaning that the application is organized as a logical grouping of related code (Richards & Ford, 2020). The table below illustrates the phase timeline and the major components implemented in each phase.

Table 1

Project Phases and Timeline

Timeline and Phases	Phase Description and Major Component
Phase 1 Backend Setup Weeks 1-2	This phase consisted of setting up the environment and Docker; gathering the data, developing the `cfr_downloader.py` script to download the MSHA data; and developing the `build_graph.py` script to chunk, vectorize, and graph the downloaded data.
Phase 2 Retrieval Tools Setup Weeks 3-4	This phase sets up the retrieval tools, such as `tools/vector.py` for vector search and `tools/cypher.py` for graph query. It also implemented `llm.py` for LLM connection and initialized the FastAPI implementation.

Phase 3 Agent & Tool Testing Weeks 5-6	This phase tested the retrieval tools. It implements the APH-IF parallel engine and fusion engine. Initialize frontend, `bot.py`.
Phase 4 Testing, Optimization, & Deployment Week 7-8	This phase finalizes the Streamlit user interface, `bot.py`, performs end-to-end testing, optimizes, and deploys the app. It also cleans/retests the code, finishes testing the FastAPI integration, and finishes project documentation.

Note: The table describes the project's phases, related major components, and timeline. From "Module 2 Capstone Milestone: Project Proposal" by Ricciardi (2025a) modified.

Modules 3 and 4: MRCA APH-IF Architecture

MRCA APH-IF is designed on a Microservice Backends-for-Frontends (BFF) architecture. The Microservice BFF architecture adds an abstraction layer (e.g., an API endpoint) between the frontend UI and the backend business logic, turning the backend and the frontend into microservices (Abdelfattah & Cerny, 2023). This design allows the application to be more secure, resilient, and maintainable. The APH-IF system, which is the core of MRCA, resides in the backend.

Architecture Overview

The frontend and backend microservices are connected via an HTTP RESTful API built with FastAPI. The frontend service is built with Streamlit, which captures user prompts and displays the final responses. It manages the user session, UI, and conversation history. The frontend's primary code resides in the `bot.py` file. The Backend service manages the entire data retrieval and answer generation process using the APH-IF system. See Figure 2 for the MRCA APH-IF architecture diagram illustrating the project architecture.

APH-IF - Advanced Parallel HybridRAG -Intelligent Fusion

The APH-IF is a novel RAG system that, unlike the regular HybridRAG technique, which chooses, using an AI agent, between VectorRAG for semantic search and GraphRAG for

traversal search, APH-IF uses a parallel engine to execute both searches concurrently for every user query and then fuses the results using a fusing engine to generate a response. The system is divided into three components, which are the parallel retrieval engine for parallel searches, a context fusion engine for combining retrieval results using algorithms, and a hybrid template system used to generate a final response based on the fused results. See Figure 3 for a class diagram illustrating the entire backend and APH-IF processes.

MRCA APH-IF is also designed to test the APH-IF system, in addition to the Intelligent Fusion (IF) strategy, referred to as Advanced Hybrid, the fusion context engine provides three other fusion strategies for testing purposes, including Weighted Linear, Max Confidence, and Adaptive fusion. Additionally, the template system (`hybrid_templates.py`) provides five different templates for final response generation, including Research-based, Regulatory Compliance, Basic Hybrid, Comparative Analysis, and Confidence Weighted. Moreover, a sophisticated metric system is used to measure the confidence and quality of retrieved data and LLM responses. Metrics such as fusion quality scores, regulatory quality scores, length scores, vector weight, graph weight, confidence level mapping, etc. These metrics are used throughout the system to make decisions and test reporting purposes.

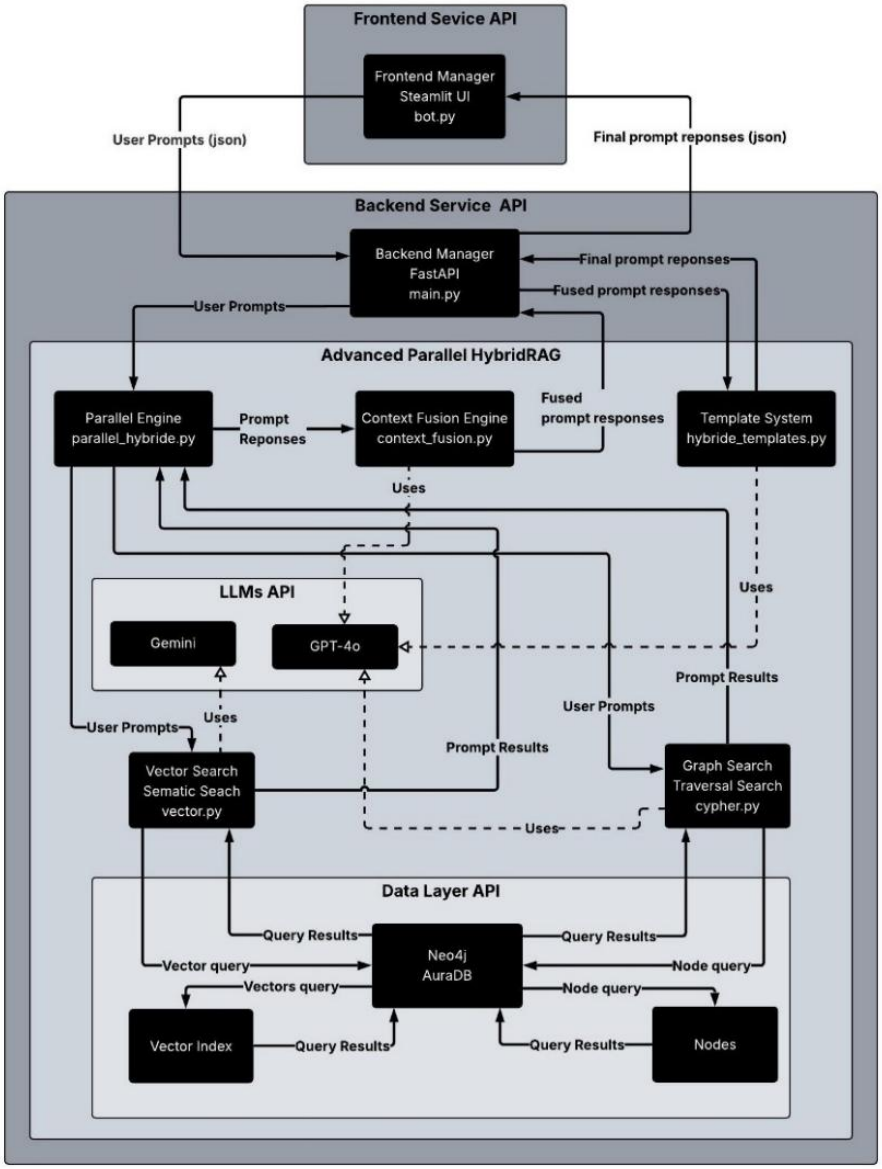
MRCA APH-IF Agentic Architecture

Additionally, the APH-IF provides a set of data retrieval tools, as illustrated in Figure 3. These tools allow the LLM GPT-4o to act as an agent by performing parallel semantic and graph-based traversal searches. Additionally, the combination of the parallel engine, fusion engine, and the template system using a set of confidence and quality metrics allows the LLM to provide accurate and reliable responses and significantly reduces the risk of hallucinations. Furthermore, the Microservice BFF architecture provides MRCA APH-IF with modular

cohesion and low coupling through frontend and backend services, allowing the system to be scalable, maintainable, and resilient. Finally, the application architecture meets all the quality attributes that should be part of any software architecture mentioned by Keeling (2017), such as modifiability, testability, availability, performance, and security.

Figure 2

MRCA Architecture Diagram



Note: The diagram illustrates the MRCA architecture with its core components and their data/control flow. From “Module 3 Capstone Milestone: Software Design” by Ricciardi (2025b).

Module 5: Fault-tolerance

The primary goal of fault-tolerance systems is to increase a software application's readability, robustness, and dependability despite internal faults or failures (Solouki et al., 2024). As an AI-powered system using a novel RAG system, MRCA APH-IF presents unique challenges for meeting fault-tolerance goals. In addition to the challenges that any software application encounters, MRCA APH-IF must overcome challenges such as non-deterministic outputs, data dependencies, and the risk of AI hallucinations. These unique challenges are directly handled by the APH-IF system, which grounds the LLM responses by using parallel retrieval and context fusion processes with a sophisticated set of metrics evaluating at each state the confidence and quality of the retrieved data and LLM-generated responses. Figure 4 - MRCA APH-IF Machine State Diagram illustrates well how this process is handled within the state of different APH-IF components’ object state.

Circuit Breaker

Additionally, at the center of the application fault-tolerant system is the circuit breaker, see Figures 3 and 4, which is implemented through the `CircuitBreaker` class in the `backen/circuit_breaker.py` file. The class monitors the health of external dependencies, which include the API connections to the Neo4j database, OpenAI LLM, and Google LLM. It also handles failure thresholds, timeouts, and recovery-testing mechanisms.

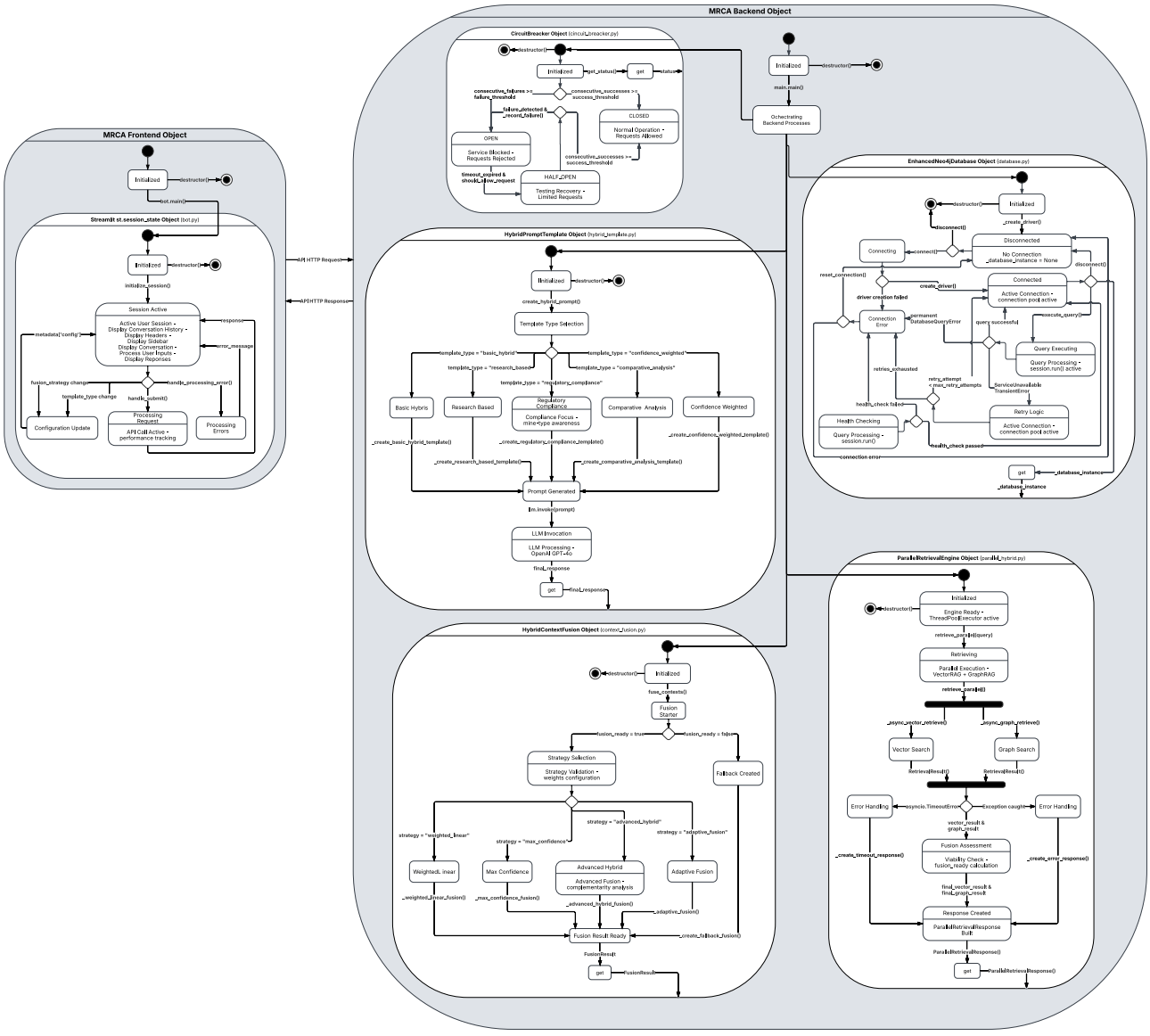
Health Monitoring and Other Fault Tolerance Components

MRCA APH-IF implements a multi-layered health monitoring system. The system monitors the FastAPI backend exposes dedicated health check endpoints (`/health`). The

individual Docker containers for the frontend and backend services also include health checks for automatic restarts if a service becomes unresponsive. In addition to the health monitoring system, the application integrates graceful degradation and fallbacks for tools, data validation, and persistent sessions using the `Pydantic` library for incoming requests and rejects any that are malformed. These fault-tolerance measures allow MRCA APH-IF to be a robust and reliable application that delivers valuable functionality and a seamless user experience.

Figure 5

MRCA APH_IF Machine State Diagram



Note: The diagram illustrates the runtime state of various frontend and backend MRCA APH-IF objects. From "Module 4 Capstone Milestone: Software Project Plan" by Ricciardi (2025c).

Modules 6 and 7: Testing Plan and Software Configuration Management

Similar to the fault-tolerance goals, the nature of MRCA APH-IF as an AI-powered application using a novel RAG system possesses unique challenges (non-deterministic outputs, data dependencies, and the risk of AI hallucinations) when designing a testing plan and a Software Configuration Management (SMC) plan for the project.

Table 2

Test Cases and SCM Controls

Test and Risk Scenario	SCM Control	Metrics	Benefit / Justification
Test case 1 - Regulatory Citation Retrieval - Component Integration Test Faulty vector embedding and graph retrieval	Implement `feature/*` branch control and control the configuration of the retrieval process. A failure will be a malformed citation and a MASH regulation section missing	≥ 95 % passage retention; p95 latency < 120 ms	This test verifies how the integrated components of the system work together (Das, 2024). It verifies the quality of the retrieval process based on the APH-IF system
Test case 2 - Multi-Domains Query - Component Integration Test Faulty vector embedding and graph query results, and intelligent fusion (`hybrid_fusion`).	Implements PR checklist, latency, and integration validity. Controls the configuration of the fusion validity across multiple MSHA regulatory domains.	Graph DB latency < 150 ms `fusion_quality` score ≥ .70	Prevents runtime failures; same rationale as test case 1. Test if the integrated components of the system fail when they are working together, and measure the quality of the fusion.
Test case 3 - Reliability Under Degraded External Services - End-to-End Testing API's faulty communication that may cause cascading faults	Implements circuit-breaker and hot-hot-fix branch control configurations. Test uses a chaos testing approach. It tests the circuit breaker functionality, on/off states, retry/backoff, and user messaging when the Neo4j database, LLM providers, or network links or API fail	Graceful 503; breaker opens ≤ 5 fails.	Helps to improve system resilience in microservices architecture by using circuit breakers to control issues related to network instability and unresponsive services (Krishna, 2023).
Test case 4 - Reliability, Fault-Injection Test, and Unit Tests Latency under heavy load	Control CI. The test uses chaos scenario fault control. It tests the performance of the system under concurrent load by injecting faults	p95 < 1.2 s; success ≥ 99 % LLM's response time should be 10–35s	Chaos testing helps to identify latent faults, improving the system's ability to respond to overcrowding, network delay, and service outages (Chintale et al., 2023).
Test case 5 - Confidence Score – Overall Architecture Evaluation (AHP) Hallucinations, off-topic prompts	Implements commit history control configuration by using `git revert`. It uses Hallucination tests by inputting 50 off-topic prompts. It tests if the system detects unsupported prompts and checks response confidence outputs	OOS precision ≥ 95 %; hallucination ≤ 5 % ; rollback ≤ 5 min	Allows for rapid recovery from hallucinations and aligns with HaluBench benchmark protocol for handling hallucinations (Ravi et al., 2024).

Test case 6 - Dependency Vulnerabilities and Security – Audits	Implements audit configuration control. The Dependabot tool is used to block vulnerable libraries (Dependencies).	Dependabot reports a vulnerable dependency	Main branch merge protection on reported dependency vulnerability. Allows for secure production code delivery. Protects production code.

Note: The table describes the test cases, their related SCM control, expected test output metric, and their benefits/justification. From "Module 7 Capstone Milestone: Software Configuration Management" by Ricciardi (2025d).

Testing Plan

MRCA APH-IF testing plan combines white-box and black-box methods. It includes unit tests to verify the functionality of code pieces; component integration tests to evaluate the core components; End-to-End (E2E) testing to evaluate the entire data flow and the system functionality from the UI input/output to the API calls; reliability and fault-injection testing to test the robustness and latency of the system; and the overall architecture evaluation score of the system against the Architecturally Significant Requirements (ASRs) of the project. Five use cases are used as the basis of the testing plan; these use cases are illustrated in Table 2. The table also illustrates how these use test cases are related to various SCM controls.

Software Configuration Management

The MRCA APH-IF SCM plan is adapted from the IEEE 828-2012 standard (IEEE, 2012) to serve the unique needs of the application AI-power and novel RAG systems. The SCM methodology is tailored for a solo-developer team; the entire SCM process was built around Git and GitHub for version control, change management, and automated testing. The project GitHub repository can be found here: https://github.com/Omegapy/MRCA-Advanced-Parallel-HybridRAG-Intelligent-Fusion. A GitFlow-Lite branching strategy was implemented to balance the needs for a rapid, 8-week development cycle using the iterative Kanban methodology and a solo-developer team. See Table 2 for a description of the SCM controls and their relation to

different tests. MRCA APH-IF SCM process supports the MRCA application throughout its software life cycle, helping it to remain maintainable, stable, and secure.

Conclusion

MRCA AHP-IF, as shown in this report, is a quick, reliable, and accurate tool for querying MSHA regulations. Its novel RAG system, AHP-IF, which uses parallel semantic and graph-based searches combined with an intelligent fusion system, is a stepping stone in RAG technology, reducing AI hallucinations and improving retrieval generation. The application was developed in an 8-week timeline using the Kanban methodology. It features microservices BFF architecture, with a containerized FastAPI backend and Streamlit frontend, making the system secure, scalable, maintainable, and resilient. The testing plan, which included unit tests, fault-injection, E2E testing, and integration testing, helped build the system's robustness, reliability, and resilience. Its SMC plans, managed via Git/GitHub using the GitFlow-Lite branching strategy, ensured that the codebase is stable, maintainable, and easily scalable for a solo-developer team. Ultimately, the MRCA APH-IF is more than just a capstone project and a reliable tool for querying MSHA regulations; it is a powerful proof-of-concept for the APH-IF technology that can potentially be applied to other fields where AI generation reliability and accuracy are paramount, including the legal, medical, and financial sectors.

References

Abdelfattah, A., & Cerny, T. (2023). *Filling the gaps in microservice frontend communication: Case for new frontend patterns*. In *Proceedings of the 13th International Conference on Cloud Computing and Services Science* (CLOSER 2023), 1, 184–193, SciTePress. https://doi.org/10.5220/0011812500003488

Chintale, P., Pandiyan, A., Chaudhari, M., Chigurupati, M., Desaboyina, G., & Malviya, R. K. (2023). *Serverless chaos engineering: A framework for fault injection and resiliency testing in AI-powered cloud workflows*. Journal of Harbin Engineering University, *44*(12), 1577-1584. https://harbinengineeringjournal.com/index.php/journal/article/view/3460

Das, S. (2024, November 4). *Integration testing and unit testing in the age of AI*. Aviator Blog. https://www.aviator.co/blog/integration-testing-and-unit-testing-in-the-age-of-ai/

IEEE. (2012). *IEEE standard for configuration management in systems and software engineering* (IEEE Std 828-2012). https://doi.org/10.1109/IEEESTD.2012.6170935

Keeling, M. (2017). Chapter 5: Dig for architecturally significant requirements. *Design it! From programmer to software architect*. Pragmatic Bookshelf. ISBN-13: 978-1-680-50209-1

Krishna, H. (2023, October 3). *What is circuit breaker in microservices?* SayOne Tech Blog. https://www.sayonetech.com/blog/circuit-breaker-in-microservices/

Ravi, S. S., Mielczarek, B., Kannappan, A., Kiela, D., & Qian, R. (2024, July 11). *Lynx: An open-source hallucination evaluation model* (arXiv:2407.08488) [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2407.08488

Ricciardi (2025a, June 15). *Module 2 capstone milestone: Project proposal* [Student Essay]. CSC480 Capstone Computer Science. CSU Global.

Ricciardi (2025b, June 29). *Module 3 capstone milestone: Software design* [Student Essay]. CSC480 Capstone Computer Science. CSU Global

Ricciardi (2025c, July 6). *Module 4 capstone milestone: Software design* [Student Essay]. CSC480 Capstone Computer Science. CSU Global

Ricciardi (2025d, July). Module 7 capstone milestone: Software configuration management [Student Essay]. CSC480 Capstone Computer Science. CSU Global

Richards, M., & Ford, N. (2020). Chapter 4: Capstone milestone: Software project plan. *Fundamentals of software architecture: An engineering approach (Softcover)* (pp. 39-56). O'Reilly Media. ISBN-13: 978-1-492-04345-4

Solouki, M. A., Angizi, S., & Violante, M. (2024, April 16). *Dependability in embedded systems: A survey of fault tolerance methods and software-based mitigation techniques* (arXiv:2404.10509v1) [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2404.10509

Xu, Z., Cruz, M. J., Guevara, M., Wang, T., Deshpande, M., Wang, X., & Li, Z. (2024, July 11). Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering. *ACM Digital Library*, SIGIR '24: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, p.2905-2909. https://doi.org/10.1145/3626772.366137