BURAQ

AI-Powered Gmail Assistant

AI407 Artificial Intelligence Capstone Lab

Project Report

WARDAH HAYA
2022622
AI

1. Executive Summary

Buraq is an intelligent, AI-powered Gmail assistant developed as a capstone submission for the AI407 Artificial Intelligence Lab course. The project demonstrates a complete, production-grade AI system lifecycle, from problem framing and data ingestion through multi-agent orchestration, security hardening, deployment, and post-production feedback analysis.

Rather than requiring users to manually navigate their inboxes, Buraq allows natural language interaction: a user can ask questions such as "Is there anything urgent today?", "Did a recruiter reply to my application?", or "Draft a response to this email" and the system will reason over real email data, retrieve relevant context, and take the correct action.

The project is structured as a lab-by-lab progression, with each module demonstrating a distinct engineering capability. The result is a working prototype that covers the full industrial AI pipeline.

2. Project Objectives

The primary objective of Buraq is to reduce the cognitive overhead of email management through a conversational AI agent. The specific goals are:

- Replace manual inbox sorting with natural language queries
- Detect urgent messages, reply to requests, spam, and recruiter outreach automatically
- Enable grounded, retrieval-augmented responses using a local knowledge base
- Draft and send emails only after explicit human confirmation
- Demonstrate all core components of an industrial AI system within a single capstone project
- Satisfy the academic requirements of AI407 across all lab submissions

3. System Architecture

Buraq is architected as a modular, multi-layer AI system. The diagram below describes the major layers and their roles:

Layer	Component	Technology / File
Data & Grounding	Email knowledge base (RAG)	ingest_data.py + Chroma vector store
Agent Core	Tool-calling reasoning loop	graph.py + LangGraph
Multi-Agent	Researcher & Analyst roles	multi_agent_graph.py + agents_config.py
Persistence	State saving & HITL pause/resume	persistence_test.py + approval_logic.py

Layer	Component	Technology / File
Security	Guardrails & jailbreak protection	secured_graph.py + guardrails_config.py
API	REST endpoint for chat/stream	main.py + FastAPI
Frontend	User UI & feedback capture	app.py + Streamlit + SQLite
Packaging	Containerised deployment	Dockerfile + docker-compose.yaml
CI/CD	Automated quality gates	.github/workflows/main.yml
Evaluation	Scored test suite	run_eval.py + lab7_evaluation.py

3.1 Data Ingestion and RAG Pipeline

The file ingest_data.py reads raw email and project data from the Initial_Data/ directory, chunks and embeds the content, and stores it in a Chroma vector database. At query time, the agent performs semantic retrieval to ground its responses in actual email content, preventing hallucination and ensuring accuracy.

3.2 LangGraph Agent Loop

The core reasoning loop is implemented in graph.py using LangGraph. The graph manages conditional routing between tool-calling steps, ensuring the agent selects the correct tool for each user intent and terminates cleanly when a final answer is ready. The available tools are defined in tools.py and cover all core email operations.

3.3 Multi-Agent Orchestration

A more advanced version of the system, implemented in multi_agent_graph.py, introduces a two-agent architecture: a Researcher agent that retrieves and surfaces relevant email information, and an Analyst agent that reasons over that information to produce a final response. Agent roles and personas are defined in agents_config.py and agent_personas.md.

3.4 Human-in-the-Loop (HITL) and Persistence

Before any email is sent, the system pauses for explicit human approval. This behavior is demonstrated in approval_logic.py. State persisted across sessions as shown in persistence_test.py, allowing interrupted workflows to be resumed without loss of context.

3.5 Security Guardrails

The secured_graph.py and guardrails_config.py files implement a security layer that detects and blocks jailbreak attempts, prevents metadata leakage, and ensures the agent cannot be manipulated into bypassing its email-sending confirmation flow.

3.6 API and Frontend

FastAPI exposes two endpoints: /chat for synchronous responses and /stream for streaming. The Streamlit frontend in app.py provides a conversational interface and captures user feedback (thumbs-up / thumbs-down) into a local SQLite database, enabling post-deployment analysis.

4. Implemented Feature Set

The following capabilities are fully implemented in the current codebase:

Feature	Description	Source File
Inbox Search	Full-text and semantic search over email data	tools.py
Date-Based Fetch	Retrieve emails by specific date or range	tools.py
Daily Summary	Auto-generated digest of the day's inbox	tools.py
Urgency Detection	Flag emails requiring immediate attention	tools.py
Reply Detection	Identify emails awaiting a response	tools.py
Spam Detection	Surface likely spam or low-priority messages	tools.py
Semantic Search (RAG)	Knowledge base retrieval for grounded answers	ingest_data.py
Draft Generation	AI-composed email drafts for user review	tools.py
Confirmed Send	Email dispatch only after explicit approval	approval_logic.py
Multi-Agent Reasoning	Researcher + Analyst collaborative pipeline	multi_agent_graph.py
State Persistence	Save and resume agent state across sessions	persistence_test.py
Guardrails	Jailbreak and metadata-leakage protection	secured_graph.py
Feedback Loop	User rating captured to SQLite	app.py
REST API	FastAPI /chat and /stream endpoints	main.py
Docker Deployment	Containerised app + data store	Dockerfile

4.1 Planned Features (PRD)

The Product Requirements Document (PRD.md) describes a broader vision. The following items are defined in the PRD but are not part of the current codebase:

- Attachment handling and summarisation workflows
- Smart scheduling and calendar integration

- Firebase / Firestore / Cloud Storage integration for scalable persistence
- Mobile-friendly frontend
- Advanced spam filtering with user-trainable classifiers

5. Lab-by-Lab Coverage

Buraq was developed incrementally across the AI407 lab curriculum. Each lab module maps directly to a demonstrable system component:

Lab	Topic	Evidence in Repo
Lab 1	Problem Framing & PRD	PRD.md, README.md
Lab 2	RAG & Vector Search	ingest_data.py, Initial_Data/
Lab 3	LangGraph Tool Use	graph.py, tools.py
Lab 4	Multi-Agent Collaboration	multi_agent_graph.py, agents_config.py, agent_personas.md
Lab 5	Persistence & HITL Approvals	persistence_test.py, approval_logic.py
Lab 6	Security Guardrails	secured_graph.py, guardrails_config.py, security_report.md
Lab 7	Evaluation & Observability	run_eval.py, lab7_evaluation.py, evaluation_report.md
Lab 8	FastAPI Serving	main.py
Lab 9	Docker Packaging	Dockerfile, docker-compose.yaml, packaging_and_cicd_report.md
Lab 10	CI/CD Quality Gates	.github/workflows/main.yml
Lab 11	Feedback & Drift Analysis	app.py (SQLite logging), drift_report.md

6. Technology Stack

Category	Technology
Language Model Framework	LangGraph (graph-based agent orchestration)
Vector Database	Chroma (local, embedded)
API Framework	FastAPI
Frontend	Streamlit
Feedback Storage	SQLite
Containerisation	Docker + Docker Compose
CI/CD	GitHub Actions
Language	Python 3

7. Security Design

Security is a first-class concern in Buraq. The system implements multiple layers of protection:

7.1 Jailbreak Prevention

secured_graph.py intercepts all user inputs before they reach the agent core. Inputs that match known jailbreak patterns or attempt to override system instructions are rejected with a safe fallback response.

7.2 Metadata Leakage Protection

The guardrails layer in guardrails_config.py ensures that internal system state, prompt templates, and configuration metadata are never exposed in agent responses, even if a user crafts a prompt specifically designed to elicit them.

7.3 Human-in-the-Loop for Destructive Actions

The send-email workflow is deliberately gated behind an explicit user confirmation step. The agent cannot dispatch an email autonomously. This HITL constraint is enforced at the graph level in approval_logic.py, making it resistant to prompt manipulation. A full write-up of the security analysis, threat model, and test results is available in security_report.md.

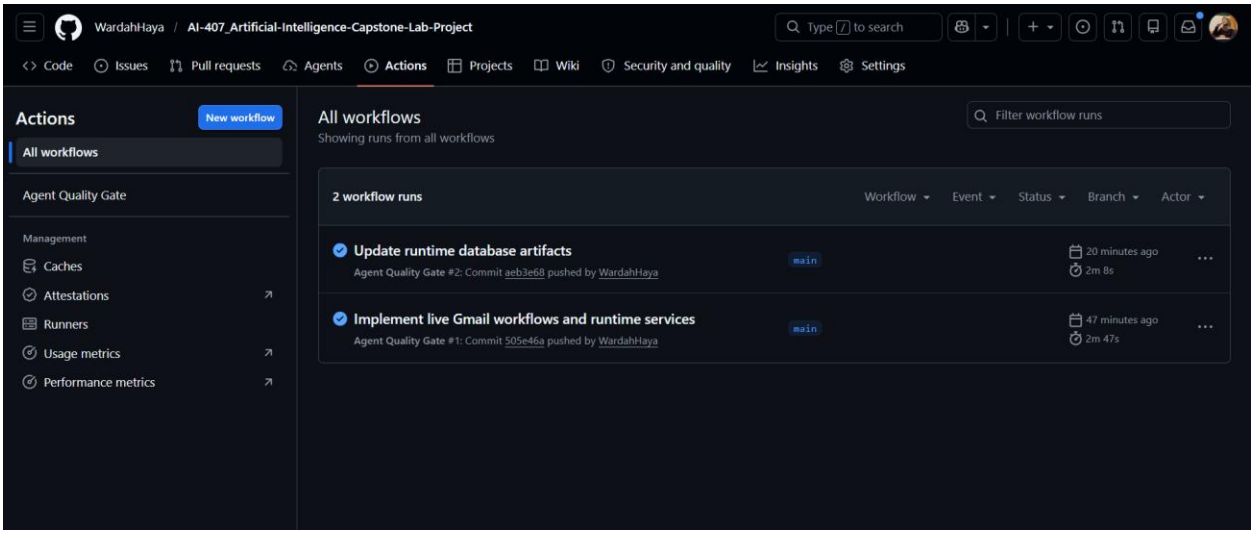
8. Evaluation & Quality Assurance

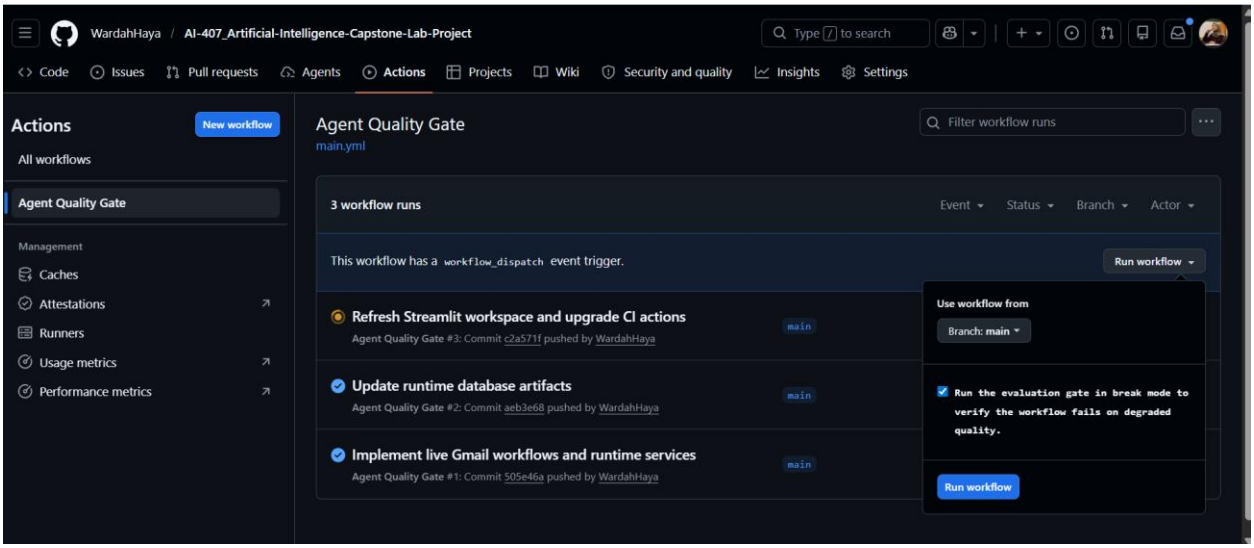
8.1 Automated Evaluation Suite

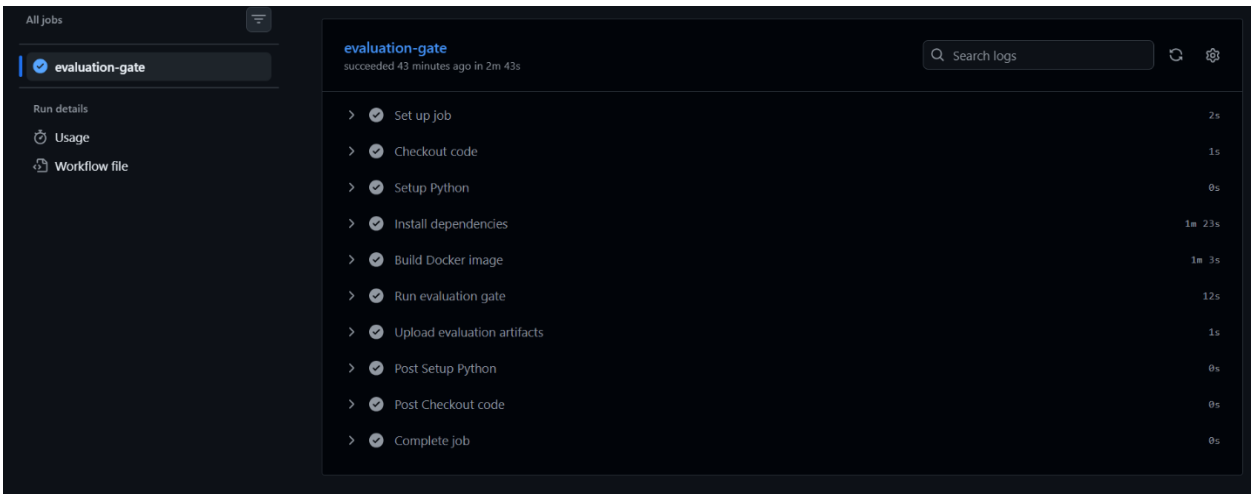
The evaluation pipeline in run_eval.py and lab7_evaluation.py runs a scored test suite against the agent. Each test case presents a simulated user query and verifies the agent's response against expected outcomes. Results are aggregated into a pass-rate score and written to evaluation_report.md.

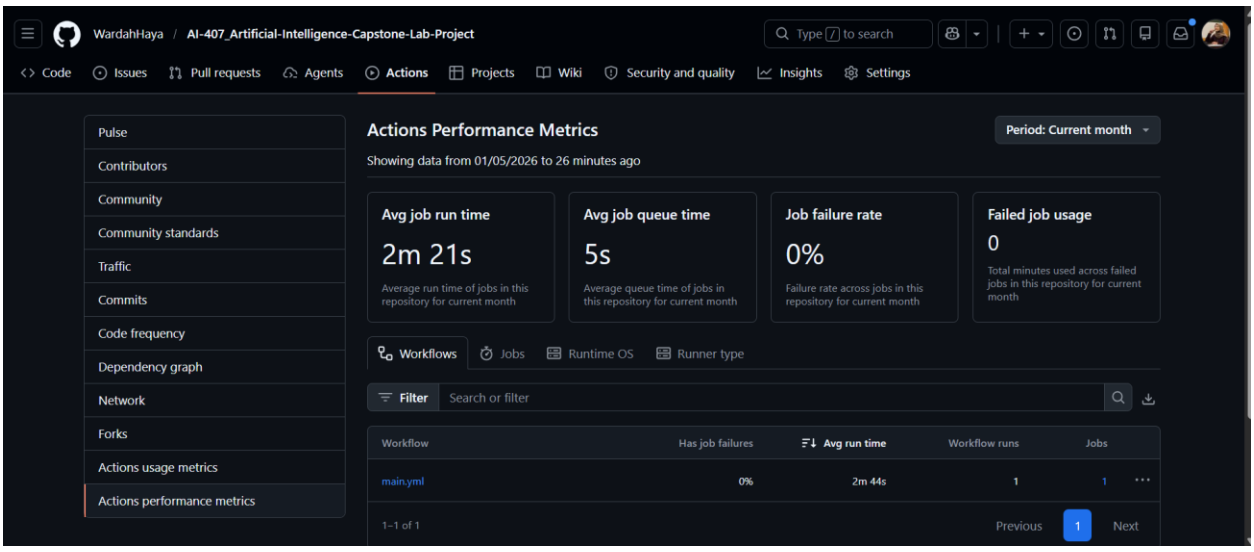
8.2 CI/CD Quality Gate

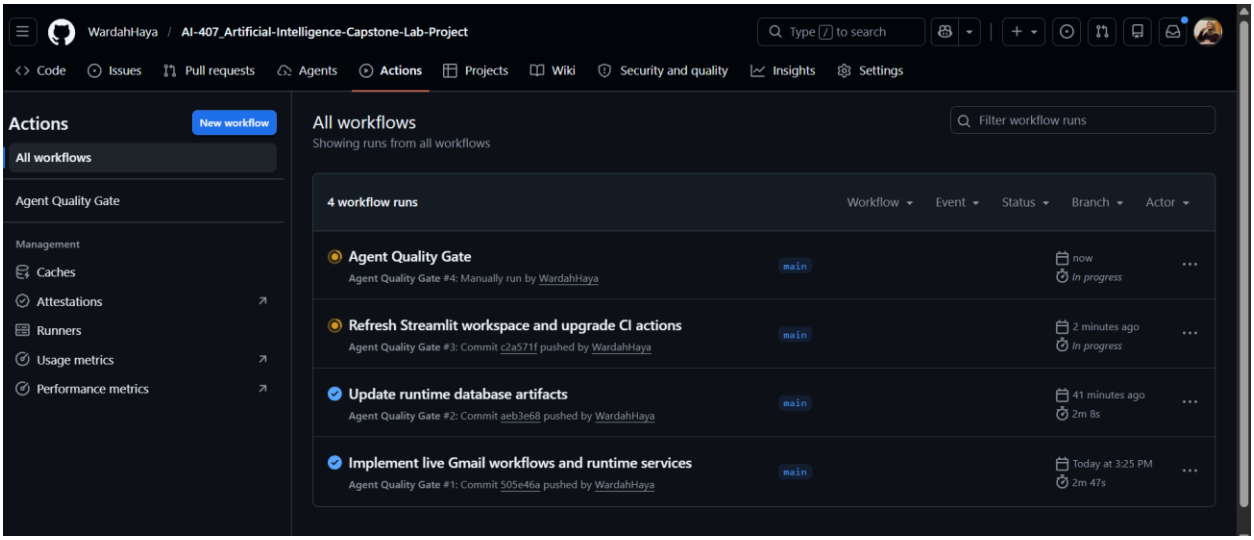
The GitHub Actions workflow in .github/workflows/main.yml automatically runs the evaluation suite on every push and pull request. Builds that fall below the minimum quality threshold are blocked, preventing regressions from reaching the main branch.

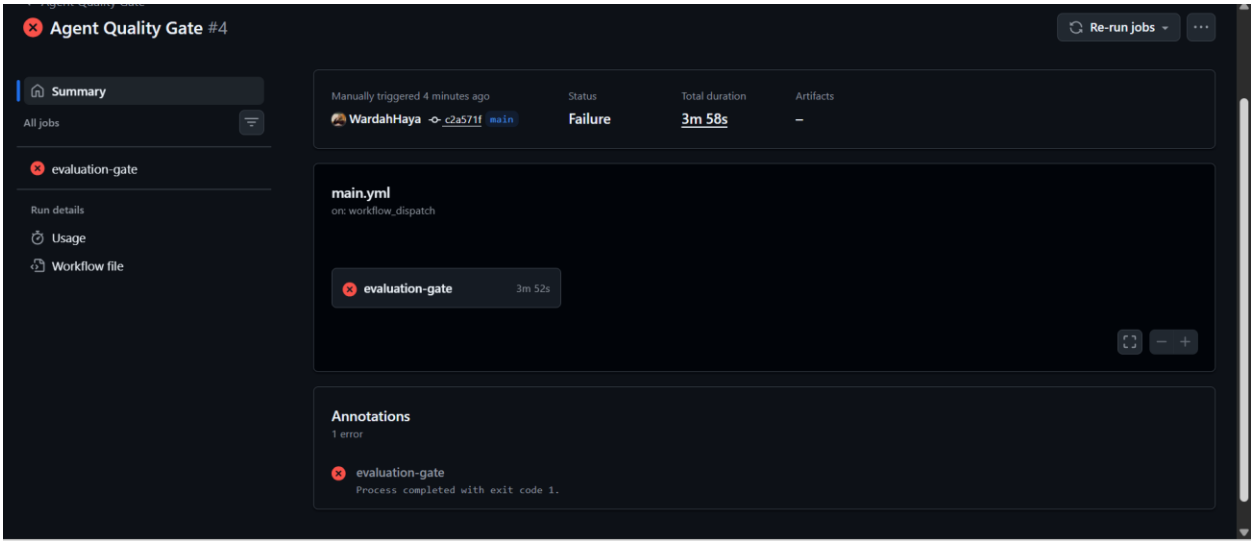


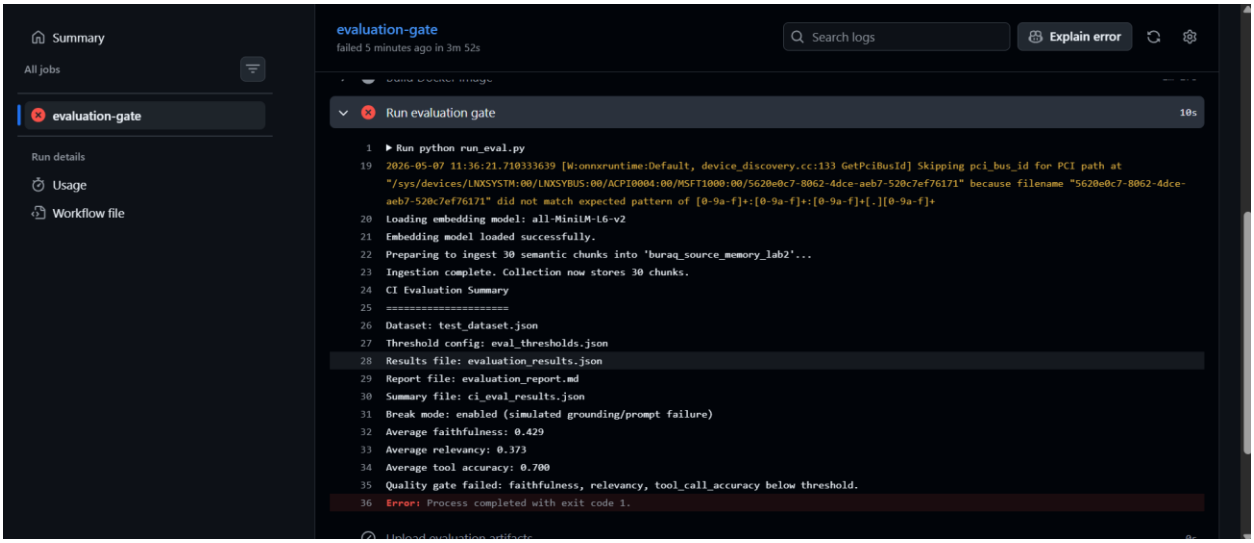

WardahHaya / AI-407_Artificial-Intelligence-Capstone-Lab-Project
Type / to search
Code
Issues
Pull requests
Agents
Actions
Projects
Wiki
Security and quality
Insights
Settings
Actions
New workflow
All workflows
Agent Quality Gate
Management
Caches
Attestations
Runners
Usage metrics
Performance metrics
Agent Quality Gate
main.yml
Filter workflow runs
3 workflow runs
Event
Status
Branch
Actor
This workflow has a workflow_dispatch event trigger.
Run workflow
Refresh Streamlit workspace and upgrade CI actions
Agent Quality Gate #3: Commit c2a571f pushed by WardahHaya
main
Update runtime database artifacts
Agent Quality Gate #2: Commit aeb3e68 pushed by WardahHaya
main
Implement live Gmail workflows and runtime services
Agent Quality Gate #1: Commit 505e46a pushed by WardahHaya
main
Use workflow from
Branch: main
Run the evaluation gate in break mode to verify the workflow fails on degraded quality.
Run workflow

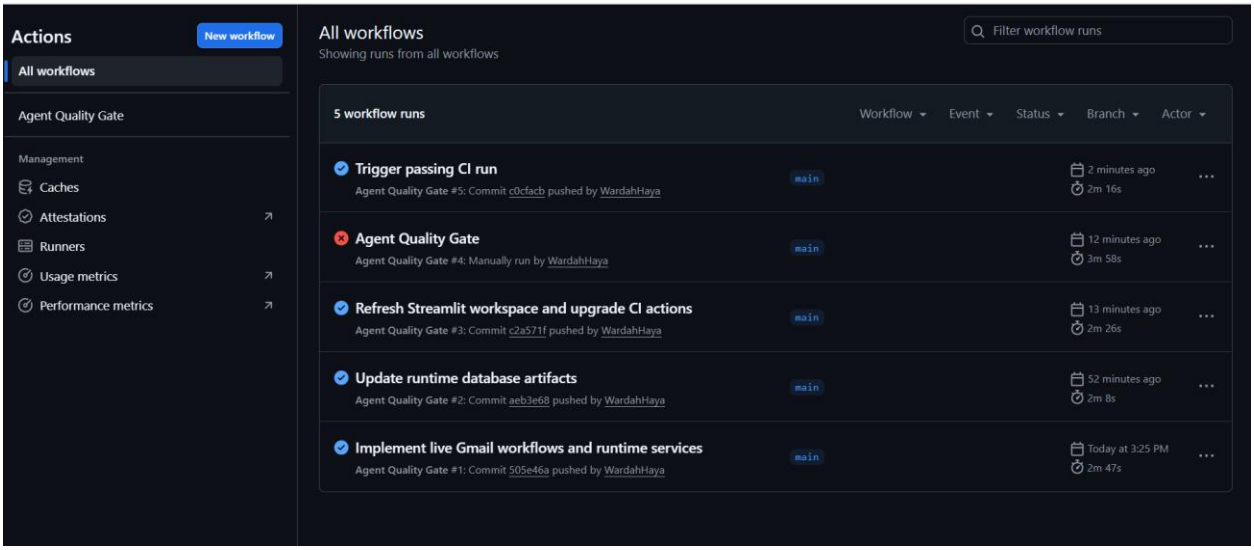

All jobs
evaluation-gate
Run details
Usage
Workflow file
evaluation-gate
succeeded 43 minutes ago in 2m 43s
Search logs
Set up job 2s
Checkout code 1s
Setup Python 0s
Install dependencies 1m 23s
Build Docker image 1m 3s
Run evaluation gate 12s
Upload evaluation artifacts 1s
Post Setup Python 0s
Post Checkout code 0s
Complete job 0s


WardahHaya / AI-407_Artificial-Intelligence-Capstone-Lab-Project
Type / to search
Code
Issues
Pull requests
Agents
Actions
Projects
Wiki
Security and quality
Insights
Settings
Pulse
Contributors
Community
Community standards
Traffic
Commits
Code frequency
Dependency graph
Network
Forks
Actions usage metrics
Actions performance metrics
Actions Performance Metrics
Period: Current month
Showing data from 01/05/2026 to 26 minutes ago
Avg job run time
2m 21s
Average run time of jobs in this repository for current month
Avg job queue time
5s
Average queue time of jobs in this repository for current month
Job failure rate
0%
Failure rate across jobs in this repository for current month
Failed job usage
0
Total minutes used across failed jobs in this repository for current month
Workflows
Jobs
Runtime OS
Runner type
Filter
Search or filter
Workflow
Has job failures
Avg run time
Workflow runs
Jobs
main.yml
0%
2m 44s
1
1
1–1 of 1
Previous
1
Next


WardahHaya / AI-407_Artificial-Intelligence-Capstone-Lab-Project
Type / to search
Code
Issues
Pull requests
Agents
Actions
Projects
Wiki
Security and quality
Insights
Settings
Actions
New workflow
All workflows
Agent Quality Gate
Management
Caches
Attestations
Runners
Usage metrics
Performance metrics
All workflows
Showing runs from all workflows
Filter workflow runs
4 workflow runs
Workflow
Event
Status
Branch
Actor
Agent Quality Gate
Agent Quality Gate #4: Manually run by WardahHaya
main
now
In progress
Refresh Streamlit workspace and upgrade CI actions
Agent Quality Gate #3: Commit c2a571f pushed by WardahHaya
main
2 minutes ago
In progress
Update runtime database artifacts
Agent Quality Gate #2: Commit aeb3e68 pushed by WardahHaya
main
41 minutes ago
2m 8s
Implement live Gmail workflows and runtime services
Agent Quality Gate #1: Commit 505e46a pushed by WardahHaya
main
Today at 3:25 PM
2m 47s


Agent Quality Gate #4
Re-run jobs
Summary
All jobs
evaluation-gate
Run details
Usage
Workflow file
Manually triggered 4 minutes ago
WardahHaya c2a571f main
Status
Failure
Total duration
3m 58s
Artifacts
–
main.yml
on: workflow_dispatch
evaluation-gate
3m 52s
Annotations
1 error
evaluation-gate
Process completed with exit code 1.


Summary
All jobs
evaluation-gate
Run details
Usage
Workflow file
evaluation-gate
failed 5 minutes ago in 3m 52s
Search logs
Explain error
Run evaluation gate
10s
1 ▶ Run python run_eval.py
19 2026-05-07 11:36:21.710333639 [W:onnxruntime:Default, device_discovery.cc:133 GetPciBusId] Skipping pci_bus_id for PCI path at "/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0004:00/MSFT1000:00/5620e0c7-8062-4dce-aeb7-520c7ef76171" because filename "5620e0c7-8062-4dce-aeb7-520c7ef76171" did not match expected pattern of [0-9a-f]+:[0-9a-f]+:[0-9a-f]+[.][0-9a-f]+
20 Loading embedding model: all-MiniLM-L6-v2
21 Embedding model loaded successfully.
22 Preparing to ingest 30 semantic chunks into 'buraq_source_memory_lab2'...
23 Ingestion complete. Collection now stores 30 chunks.
24 CI Evaluation Summary
25 ====================
26 Dataset: test_dataset.json
27 Threshold config: eval_thresholds.json
28 Results file: evaluation_results.json
29 Report file: evaluation_report.md
30 Summary file: ci_eval_results.json
31 Break mode: enabled (simulated grounding/prompt failure)
32 Average faithfulness: 0.429
33 Average relevancy: 0.373
34 Average tool accuracy: 0.700
35 Quality gate failed: faithfulness, relevancy, tool_call_accuracy below threshold.
36 Error: Process completed with exit code 1.



8.3 Post-Deployment Feedback

The Streamlit frontend captures per-response thumbs-up / thumbs-down ratings into a SQLite database. This data is analysed in drift_report.md to identify capability gaps and performance drift over time, closing the feedback loop between user experience and model improvement.

9. Deployment

Buraq is fully containerised for consistent deployment across environments:

- The Dockerfile packages the Python application and all dependencies into a single image.
- docker-compose.yaml orchestrates the application container alongside its Chroma vector store, exposing the FastAPI service on a configurable port.
- Environment variables are used for all secrets and configuration, following twelve-factor app principles.
- The CI/CD pipeline (GitHub Actions) builds, tests, and validates the Docker image on every commit.

A detailed walkthrough of the packaging process and deployment architecture is documented in packaging_and_cicd_report.md.

```
PS C:\Users\PC\Downloads\Updated capstone lab project\AI-407_Artificial-Intelligence-Capstone-Lab-Project-main> docker compose build
#1 [internal] load local bake definitions
#1 reading from stdin 787B 0.0s done
#1 DONE 0.0s

#2 [internal] load build definition from Dockerfile
#2 transferring dockerfile: 520B 0.0s done
#2 DONE 0.0s

#3 [internal] load metadata for docker.io/library/python:3.11-slim
#3 ...

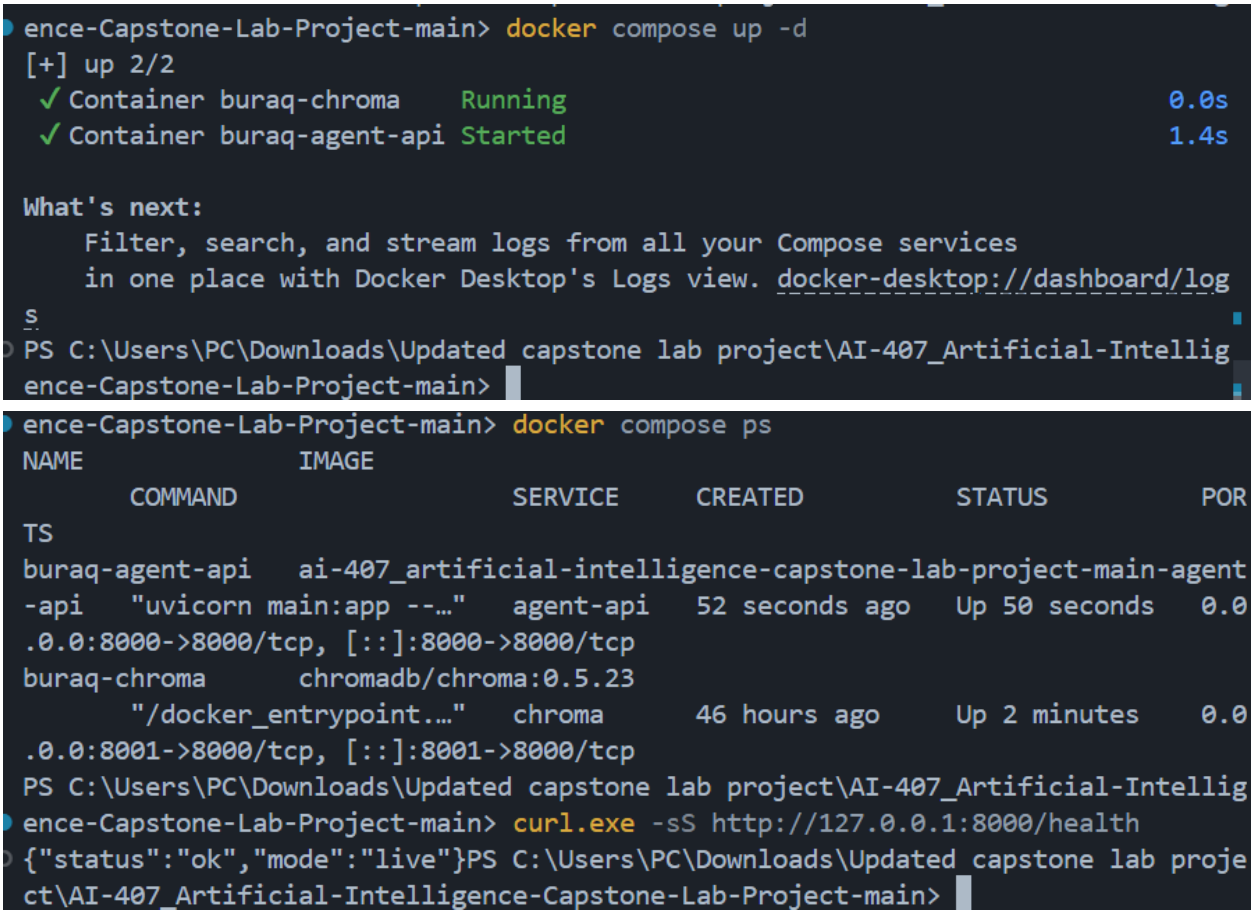
#4 [auth] library/python:pull token for registry-1.docker.io
#4 DONE 0.0s

#3 [internal] load metadata for docker.io/library/python:3.11-slim
#3 DONE 5.5s

#5 [internal] load .dockerignore
#5 transferring context: 288B done
#5 DONE 0.0s
```

```
PS C:\Users\PC\Downloads\Updated capstone lab project\AI-407_Artificial-Intelligence-Capstone-Lab-Project-main> docker compose build
#13 exporting config sha256:906824a4cab02db63ca6017b6e2693c1b87d526248296d100b6656aa3999ed93 0.0s done
#13 exporting attestation manifest sha256:49081be7b04bdfb218067a1d138750a06a2976d2a7c152973e569ea17fa18456 0.0s done
#13 exporting manifest list sha256:161d25f4a2d82f8953b6f645b93365512af0c9335a4251debd2bc4eb990f3439
#13 exporting manifest list sha256:161d25f4a2d82f8953b6f645b93365512af0c9335a4251debd2bc4eb990f3439 0.0s done
#13 naming to docker.io/library/ai-407_artificial-intelligence-capstone-lab-project-main-agent-api:latest done
#13 unpacking to docker.io/library/ai-407_artificial-intelligence-capstone-lab-project-main-agent-api:latest 0.1s done
#13 DONE 0.4s

#14 resolving provenance for metadata file
#14 DONE 0.0s
[+] build 1/1
 ✓ Image ai-407_artificial-intelligence-capstone-lab-project-main-... Built 4.9s
PS C:\Users\PC\Downloads\Updated capstone lab project\AI-407_Artificial-Intelligence-Capstone-Lab-Project-main>
```



10. Current Status and Limitations

10.1 What Works

The project is a fully functional capstone prototype. All core email management workflows, reading, summarising, urgency detection, drafting, and confirmed sending are operational. The multi-agent pipeline, security guardrails, evaluation suite, CI/CD gate, and Docker deployment are all implemented and demonstrable.

10.2 Known Limitations

- Attachment handling, scheduling, and Firebase integration are described in the PRD but not yet implemented.
- The Streamlit UI is functional but not designed for production-scale user load.
- The evaluation dataset is synthetic; real-world performance may vary.

11. Feedback Analytics and Self-Improvement Loop

This lab extends an intelligent email assistant by implementing a feedback-driven analytics and self-improvement pipeline. The system collects user feedback after each assistant response, stores it in a persistent SQLite database, analyzes failure patterns, and automatically improves the runtime prompt using historical feedback.

1. Feedback Collection System
   A feedback mechanism was integrated into the frontend interface to let users evaluate each assistant response.

Implemented features

- Thumbs Up feedback (👍)
- Thumbs Down feedback (👎)
- Optional textual comments
- Persistent storage of feedback entries

Frontend integration

- The feedback UI is displayed after every assistant response.
- Implemented in: app.py (Line 1169), app.py (Line 1257)
- The interface allows the user to open the “Rate this response” section, select positive or negative feedback, and save the feedback entry into the database.

2. Feedback Database
   A dedicated SQLite database persists all user evaluations.

Database file

- feedback_log.db

Database initialization

- The database and required tables are initialized automatically during application startup.
- Implemented in: app.py (Line 1951)

Stored fields

- user_input: Original user query
- agent_response: Assistant-generated response
- feedback_score: Positive or negative rating
- timestamp: Time of interaction
- thread_id: Conversation thread identifier
- message_id: Message identifier
- optional_comment: Optional user feedback

Logging implementation

- Feedback logging functionality was implemented in: app.py (Line 156), app.py (Line 175)

3. Feedback Analytics Pipeline
   An analytics script was developed to process collected feedback and generate improvement reports.

Analysis script

- analyze.py

Implemented features

- Total response counting

- Positive/negative feedback analysis
- Failure rate computation
- Identification of most problematic queries
- Failure category extraction
- Automatic prompt refinement generation

Core analysis sections

- Implemented in: analyze.py (Line 38), analyze.py (Line 128), analyze.py (Line 148), analyze.py (Line 260)

4. Runtime Self-Improvement Integration
   An automated self-improvement loop applies feedback-driven improvements to the agent at runtime.

Workflow

- Negative feedback is analyzed
- Common failure patterns are identified
- An improved runtime prompt is generated
- The improved prompt is automatically loaded into the agent

Implementation points

- Prompt generation: analyze.py (Line 78)
- Runtime prompt loading: prompt_overrides.py (Line 19), prompt_overrides.py (Line 26), graph.py (Line 43)

This ensures that improvements derived from user feedback are actively applied in future interactions.

5. Generated Deliverables
   The analytics pipeline automatically generates the following files:

- feedback_log.db — Stores all feedback
- analysis_report.md — Feedback statistics report
- drift_report.md — Drift and issue analysis
- improvement_demo.md — Before/after improvement examples
- improved_prompt.txt — Regenerated runtime prompt

6. Execution Procedure
   Step 1 — Run backend

- python -m uvicorn main:app --host 127.0.0.1 --port 8000

Step 2 — Run frontend

- streamlit run app.py

Step 3 — Use the assistant

- Interact with the assistant normally through the UI.
- After each response: open “Rate this response”, select 👍 or 👎, click Save Feedback.
- Each save inserts a feedback row into the SQLite database.

7. Running the Analytics Pipeline

- Execute: python analyze.py
- The script reads feedback_log.db, computes analytics, detects common failures, regenerates the improvement prompt, and produces updated reports.

8. Experimental Results
   Output obtained (example run)

- Total responses: 10
- Negative feedback: 6

Top 3 failed queries

- “Write a warm reply to Talent Team.”
- “What is the deadline linked to the recruiter interview email?”
- “Find the email about the evaluation rubric.”

9. Failure Analysis
   Detected failure categories

- missing_context: 3
- tone_mismatch: 2
- tool_failure: 1

Observations

- Context retrieval failures caused incomplete answers.
- Tone mismatches affected generated email quality.
- Tool execution failures impacted retrieval accuracy.

10. Improvement Demonstration
    The system automatically generated an improved runtime prompt based on historical negative feedback.

Improvement process

- Failure trends were extracted from persisted feedback.
- A prompt refinement was generated dynamically.
- The updated prompt was loaded back into runtime execution.

Demonstrated in

- improvement_demo.md

11. Conclusion

Buraq successfully demonstrates a complete, end-to-end industrial AI system within the scope of an academic capstone. The project covers every major phase of the AI lifecycle: problem framing, data grounding, agentic reasoning, multi-agent collaboration, human-in-the-loop controls, security hardening, automated evaluation, API serving, containerised deployment, and post-deployment feedback analysis.

The codebase is clean, modular, and well-documented, with each component traceable to a specific lab module. While the PRD outlines a more ambitious long-term product vision, the implemented prototype already delivers meaningful value and represents a sophisticated body of AI engineering work.

Buraq is a strong foundation for a production Gmail assistant and a comprehensive demonstration of the skills developed throughout the AI407 curriculum.