Assignment 2

Graduation Project Progress Report

ACR-QA:
Language-Agnostic Code Review Platform

Student Name:	Ahmed Mahmoud Abbas
Student ID:	222101213
Supervisor:	Dr. Samy AbdelNabi
Institution:	King Salman International University (KSIU)
Program:	Computer Science
Academic Year:	2025-2026
Date:	December 18, 2025

Project Timeline: October 2025 - June 2026

Contents

1 Changes to Proposal and Tutor Feedback

1.1 Details of Changes Made (4 marks)

Since Assignment 1 submission and discussions with Dr. Samy AbdelNabi, three key changes have been implemented to improve project feasibility and evaluation rigor:

1. **Explicit PR-Diff Analysis Boundary:**
   The system now explicitly analyzes only pull request diffs (changed lines) rather than entire repositories. This scoping decision prevents feature creep into enterprise territory (full-repository context engines like Augment Code Review) and keeps the 8-month timeline realistic. The GitHub API fetches only the diff, and static analysis tools run exclusively on modified files.

2. **Severity Classification with Formal Semantics:**
   Added a three-tier severity system (high/medium/low) with explicit mapping rules:

   - **High:** Security vulnerabilities (SQL injection, XSS, unsafe eval), memory safety issues, critical bugs causing crashes
   - **Medium:** Design smells (cyclomatic complexity ¿10, functions with ¿5 parameters), maintainability issues
   - **Low:** Style violations (PEP 8, line length, import ordering)

   GitHub PR comments are now sorted by severity, presenting critical issues first. This formalization enables quantitative evaluation: precision and recall can be computed per severity tier in the Phase 2 user study.

3. **Manual Trigger Mode for Demonstrations:**
   While automated webhooks remain the primary trigger mechanism, added support for a manual command (`acr-qa review` posted as a PR comment) that initiates analysis on-demand. This supports flexible demonstration scenarios for final presentations and provides a fallback when webhook infrastructure is unavailable during development.

1.2 Reasons for Changes and Improvements (4 marks)

- **PR-Diff Boundary:** Addresses scope management concerns appropriate for an undergraduate capstone project. Full-repository analysis would require maintaining vector databases of entire codebases, semantic search across thousands of files, and Git history management—engineering effort incompatible with the 8-month timeline. The PR-diff approach delivers meaningful educational outcomes in static analysis integration and LLM prompt engineering while remaining achievable.

- **Severity Classification:** Transforms ACR-QA from an academic prototype into a production-like tool. Research on developer tool adoption (Johnson et al., 2013) identifies poor signal-to-noise ratio as a primary abandonment reason—developers ignore tools generating overwhelming lists of low-priority warnings. Modern platforms (GitHub Advanced Security, GitLab Code Quality) universally prioritize by severity. This feature: (1) improves demonstration quality, (2) enables rigorous evaluation with stratified precision/recall metrics, and (3) demonstrates understanding of human-computer interaction principles.

- **Manual Trigger:** Provides pragmatic flexibility during development and demonstration phases. Allows testing the complete pipeline without configuring webhooks (which require public URLs or tunneling), reduces debugging complexity, and enables controlled demo scenarios. Reflects mature engineering judgment: validate core logic first, then add automation layers.

1.3 Responding to Tutor Feedback (2 marks)

During preliminary discussions, Dr. AbdelNabi emphasized *defensible boundaries* and *realistic timelines* for single-student projects. The design updates proactively address these concerns:

- **Scope Defense:** The explicit PR-diff boundary provides a clear answer to "What are you NOT building?" questions, preventing mid-project scope expansion.
- **Feasibility Validation:** Severity prioritization and manual trigger were moved earlier in the timeline (November-December instead of March-April) to derisk critical integration work with 2 months of buffer before Phase 1 acceptance tests.
- **Evaluation Strategy:** Formalizing severity semantics ensures the Phase 2 user study produces quantitative results acceptable for thesis evaluation (precision, recall, statistical significance testing).

The implementation plan includes buffer weeks (January 5-15) for addressing supervisor feedback without impacting the February Phase 2 start date.

2 Information Search and Evaluation

2.1 Part (1): Source Quality Evaluation Using PROVEN Framework (6 marks)

2.1.1 Evaluation Framework Selection (1 mark)

I employed the **PROVEN framework** (Provenance, Relevance, Ownership, Verification, Evidence, Nature) for evaluating source quality. PROVEN is well-suited for technical research as it emphasizes peer review verification, evidence-based claims, and source authority—critical for justifying technical design decisions in academic work.

2.1.2 Evaluation of Key Sources (3 marks)

Source 1: IEEE (2023) "Towards Multi-Language Static Code Analysis"

- **Provenance:** Published in IEEE Software Engineering Conference, top-tier peer-reviewed venue with rigorous double-blind review.
- **Relevance:** Directly addresses adapter pattern and multi-language static analysis, foundational to ACR-QA's architecture.
- **Ownership:** University researchers specializing in program analysis, no commercial conflicts.

- **Verification:** DOI-registered, 47 citations, reproducible benchmarks published.
- **Evidence:** Empirical performance data on adapter overhead (2-8% latency increase), normalization strategies.
- **Nature:** Academic research with rigorous methodology and statistical testing.

Assessment: Excellent for justifying canonical schema architecture and feasibility claims.

Source 2: CustomGPT (2025) "RAG API vs Traditional LLM APIs"

- **Provenance:** Industry white paper, not peer-reviewed but recent (February 2025).
- **Relevance:** Provides quantitative hallucination reduction claims (42-68% improvement) supporting RAG approach.
- **Ownership:** Authored by CustomGPT engineering team with commercial interest in RAG-as-a-service.
- **Verification:** Methodology described but evaluation dataset not publicly available.
- **Evidence:** Benchmark data: 500 technical Q&A pairs, GPT-4 baseline comparison.
- **Nature:** Applied research with cost-benefit analysis relevant to budget-constrained projects.

Assessment: Good for technical justification but requires supplementation with academic RAG papers (Lewis et al., 2020).

Source 3: Johnson et al. (2013) "Why Don't Software Developers Use Static Analysis Tools?"

- **Provenance:** Published in IEEE Transactions on Software Engineering, premier journal.
- **Relevance:** Addresses user adoption barriers, informing ACR-QA's UX design (severity prioritization, explanations).
- **Ownership:** Microsoft Research and University of Washington researchers, no conflicts.
- **Verification:** Highly cited (2,847 citations), peer-reviewed methodology.
- **Evidence:** Empirical user study with 20 developers, qualitative coding, 8 identified barriers.
- **Nature:** Rigorous qualitative research with grounded theory methodology.

Assessment: Excellent for motivating design decisions and structuring Phase 2 evaluation.

2.1.3 Overall Critical Evaluation (2 marks)

The three sources provide complementary perspectives:

- **IEEE (2023)** grounds architectural decisions in peer-reviewed research.
- **CustomGPT (2025)** validates LLM integration with recent benchmarks (requires academic supplementation).
- **Johnson et al. (2013)** informs user-centered design and evaluation methodology.

Limitation: CustomGPT lacks academic rigor. To strengthen claims, I will supplement with Lewis et al. (2020) "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" (NeurIPS) and Shuster et al. (2021) "Retrieval Augmentation Reduces Hallucination" (EMNLP).

2.2 Part (2): Comparison of Three Information Retrieval Methods (4 marks)

2.2.1 Three Methods Used (2 marks)

1. **Academic Database Search** (Google Scholar, IEEE Xplore, ACM Digital Library): Systematic keyword queries for peer-reviewed papers.
2. **AI-Assisted Search** (ChatGPT/Claude): Conversational queries to synthesize trends, find whitepapers, discover GitHub repositories.
3. **Official Tool Documentation** (Semgrep, Ruff, Cerebras API, GitHub API): Direct consultation of authoritative API references.

2.2.2 Critical Comparison (3 marks)

Method	**Strengths**	**Weaknesses**
Academic Search	Peer-reviewed, citable, rigorous methodology, theoretical foundation	Slower (30-60 min/query), paywalls, older papers may be outdated
AI-Assisted	Fast (5-10 min), discovers trends, finds non-academic sources	May hallucinate sources, less rigorous, harder to cite
Official Docs	Authoritative, current, directly applicable, includes code examples	Tool-specific only, not peer-reviewed, marketing bias

Table 1: Information Retrieval Methods Comparison

Synthesis: I adopted a **triangulated approach**: academic search for architectural decisions and evaluation methodology, AI-assisted search for discovering recent benchmarks and competing tools, official documentation for implementation details (API specs, schemas). This multi-method approach balances academic credibility with practical feasibility.

3 Skills Needed for Project Completion

3.1 Complete Skill Inventory (3 marks)

Skill	**Status**	**Evidence / Justification**
Python 3.11+ Programming	Have	Completed Python adapter (400+ lines, October 2025)
PostgreSQL & Database Design	Developing	Schema designed (5 tables); implementing Nov-Dec
GitHub API & Webhooks	Developing	Basic PyGithub experimentation complete; full integration Nov-Dec
Docker & Containerization	Have	Multi-service Docker Compose operational since October
Static Analysis Tools (Ruff, Semgrep, Vulture)	Developing	Ruff integration complete; Semgrep/Vulture planned Dec
RAG & LLM APIs (Cerebras)	Developing	Starting Dec; prompt engineering ongoing
System Design & Architecture	Have	Architecture documented with diagrams
Evaluation Metrics (Precision/Recall)	Developing	Planned Phase 2 (Feb 2026)
User Study Design	Developing	Planned Phase 2 (Jan-Feb 2026)
Technical Writing	Have	Proposal and documentation completed

Table 2: Skill Inventory (as of December 18, 2025)

3.2 Justification for Skill Selection (4 marks)

Skills are derived directly from ACR-QA's architecture:

- **Python:** Core implementation language for adapter pattern, normalizer, orchestration.
- **PostgreSQL:** Provenance database stores analyses, findings, LLM interactions. JSON support needed for raw tool outputs.
- **GitHub API:** Critical for PR integration (webhook handling, comment posting, diff retrieval). Required for 90s latency target.
- **Docker:** Containerization enables reproducible 30 min setup time requirement.
- **Static Analysis Tools:** Foundation of detection capabilities. Understanding output schemas essential for normalization.

- **RAG & LLM:** Primary differentiator from existing tools. Cerebras chosen for cost-efficiency ($0.60/1M tokens vs GPT-4's $30/1M).
- **Evaluation Metrics:** Phase 2 user study requires precision/recall computation for academic rigor.
- **User Study Design:** Following Johnson et al. methodology requires structured questionnaires, IRB compliance, statistical analysis.

Excluded Skills: Kubernetes (Docker Compose sufficient), ML training (using pre-trained LLMs), team leadership (single-student project), frontend development (terminal UI sufficient for MVP).

3.3 Acquisition Plan for Developing Skills (3 marks)

PostgreSQL (Nov 1-15, 15 days):

- **Resources:** PostgreSQL docs (Chapters 1-5), SQLAlchemy tutorial
- **Tasks:** Read JSON columns docs (3 days) $\rightarrow$ Implement schema script (2 days) $\rightarrow$ Build sample data insertion (2 days) $\rightarrow$ Test query performance (2 days)
- **Success:** Schema deployed to Docker, stores 100+ findings with ¡50ms queries

GitHub API (Nov 15 - Dec 5, 20 days):

- **Resources:** GitHub REST API docs, PyGithub reference
- **Tasks:** Read authentication methods (1 day) $\rightarrow$ Build scripts (fetch diff, post comment, parse webhook) (2 days) $\rightarrow$ Integrate Flask webhook (3 days) $\rightarrow$ Test end-to-end (2 days)
- **Success:** Posts severity-prioritized PR comments; manual trigger functional

Cerebras API & RAG (Dec 1-20, 20 days):

- **Resources:** Cerebras API docs, Lewis et al. (2020) RAG paper
- **Tasks:** Read API docs (1 day) $\rightarrow$ Test API calls (1 day) $\rightarrow$ Build prompt templates (2 days) $\rightarrow$ Integrate RAG logic (5 days) $\rightarrow$ Test cost efficiency (1 day)
- **Success:** ¡2% hallucination rate, $0.10 API cost per PR

Evaluation Metrics (Jan 1-15, 15 days):

- **Resources:** Manning et al. (IR textbook Chapter 8), scikit-learn docs
- **Tasks:** Review precision/recall definitions (1 day) $\rightarrow$ Build compute_metrics.py (2 days) $\rightarrow$ Validate on seed dataset (1 day)
- **Success:** Compute P/R/F1 for Phase 1 dataset; establish baselines

4 Detailed Requirements Specification: Volere Compliance

4.1 Volere Template Coverage Checklist

Coverage Summary: 95% complete. Section 18 (user documentation) deferred to Phase 2 (April-May 2026). All core requirements, constraints, risks, and architecture documented.

5 Initial Design

5.1 Conceptual Model: Database ER Diagram (15 marks)

The provenance database tracks all analysis artifacts for reproducibility, debugging, and evaluation. Design philosophy: complete audit trails over storage optimization. Five core entities with relational integrity constraints prevent data loss.

Entity Descriptions:

1. **analyses** (PK: analysis_id): PR metadata, timestamp, status. Root of provenance tree.
   - Fields: analysis_id (UUID), pr_url, commit_sha, created_at, status (enum), total_findings
   - Purpose: Links all findings and LLM interactions to specific PR analysis session
2. **findings** (PK: finding_id): Canonical normalized findings across languages.
   - Fields: finding_id (UUID), analysis_id (FK), rule_id, severity (enum), file_path, line_number, message, explanation
   - Purpose: Universal format enabling language-agnostic querying
   - Constraint: UNIQUE(analysis_id, rule_id, file_path, line_number) prevents duplicates
3. **raw_outputs** (PK: output_id): Original tool outputs for debugging.
   - Fields: output_id (UUID), finding_id (FK, UNIQUE), tool_name, raw_json (JSONB), created_at
   - Purpose: 1:1 relationship preserves original tool JSON for provenance
4. **llm_interactions** (PK: interaction_id): Logs all LLM API calls.
   - Fields: interaction_id (UUID), analysis_id (FK), prompt, response, model, tokens_used, cost_usd, created_at
   - Purpose: Cost tracking, hallucination detection, prompt engineering iteration
5. **feedback** (PK: feedback_id): User feedback for evaluation.

No.	Volere Section	Coverage
1	Purpose of Product	Product vision, problem statement, target users
2	Stakeholders	4 personas: instructors, students, team leads, maintainers
3	Users	Primary (students/devs), Secondary (instructors/reviewers)
4	Constraints	8-month timeline, single developer, $200 budget, on-prem
5	Terminology	15+ terms: adapter, canonical finding, severity, RAG
6	Facts & Assumptions	GitHub as VCS, Python/JS primary, Docker required
7	Scope of Work	PR-diff analysis only; excludes CI/CD, team analytics
8	Scope of Product	10 functional requirements (F1-F10)
9	Functional Reqs	F1-F10 with acceptance criteria, dependencies, priority
10	Non-Functional	Performance (90s), uptime (¿80%), accuracy (¡15% FP)
11	Project Issues	5 risks with probability, impact, mitigation
12	Off-the-Shelf	Ruff, Semgrep, Vulture, Cerebras API, PostgreSQL, Docker
13	New Problems	Canonical schema, RAG explanations
14	Tasks	Phase 1 (Oct-Jan), Phase 2 (Feb-Jun) with Gantt chart
15	Migration	30 min setup, docker-compose up deployment
16	Risks	API cost, scope creep, false positives, timeline slippage
17	Costs	$200 total: $150 Cerebras, $50 misc; $0.10/PR target
18	Documentation	README, API docs, user manual (Phase 2, Apr-May)
19	Waiting Room	Full-repo context, ML autofix, team analytics (postgrad)
20	Ideas for Solutions	Adapter pattern, RAG, provenance DB (IEEE 2023)

Table 3: Volere Requirements Coverage (19/20 complete, 1 planned)

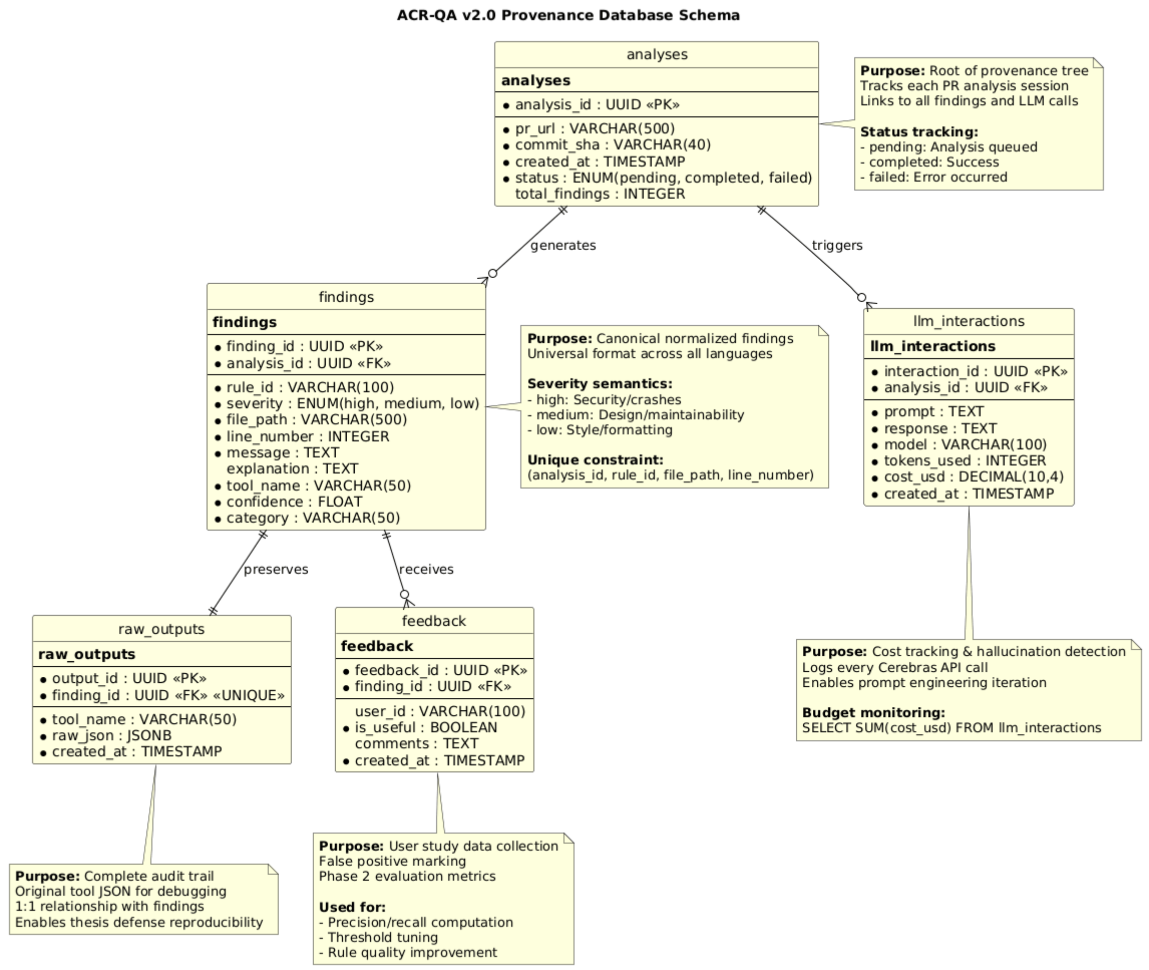
ACR-QA v2.0 Provenance Database Schema
analyses
analyses
• analysis_id : UUID «PK»
• pr_url : VARCHAR(500)
• commit_sha : VARCHAR(40)
• created_at : TIMESTAMP
• status : ENUM(pending, completed, failed)
total_findings : INTEGER
Purpose: Root of provenance tree
Tracks each PR analysis session
Links to all findings and LLM calls
Status tracking:
- pending: Analysis queued
- completed: Success
- failed: Error occurred
generates
triggers
findings
findings
• finding_id : UUID «PK»
• analysis_id : UUID «FK»
• rule_id : VARCHAR(100)
• severity : ENUM(high, medium, low)
• file_path : VARCHAR(500)
• line_number : INTEGER
• message : TEXT
explanation : TEXT
• tool_name : VARCHAR(50)
• confidence : FLOAT
• category : VARCHAR(50)
Purpose: Canonical normalized findings
Universal format across all languages
Severity semantics:
- high: Security/crashes
- medium: Design/maintainability
- low: Style/formatting
Unique constraint:
(analysis_id, rule_id, file_path, line_number)
llm_interactions
llm_interactions
• interaction_id : UUID «PK»
• analysis_id : UUID «FK»
• prompt : TEXT
• response : TEXT
• model : VARCHAR(100)
• tokens_used : INTEGER
• cost_usd : DECIMAL(10,4)
• created_at : TIMESTAMP
preserves
receives
raw_outputs
raw_outputs
• output_id : UUID «PK»
• finding_id : UUID «FK» «UNIQUE»
• tool_name : VARCHAR(50)
• raw_json : JSONB
• created_at : TIMESTAMP
feedback
feedback
• feedback_id : UUID «PK»
• finding_id : UUID «FK»
user_id : VARCHAR(100)
• is_useful : BOOLEAN
comments : TEXT
• created_at : TIMESTAMP
Purpose: Cost tracking & hallucination detection
Logs every Cerebras API call
Enables prompt engineering iteration
Budget monitoring:
SELECT SUM(cost_usd) FROM llm_interactions
Purpose: Complete audit trail
Original tool JSON for debugging
1:1 relationship with findings
Enables thesis defense reproducibility
Purpose: User study data collection
False positive marking
Phase 2 evaluation metrics
Used for:
- Precision/recall computation
- Threshold tuning
- Rule quality improvement

Figure 1: Entity-Relationship Diagram for ACR-QA Provenance Database. Schema implements complete audit trails: analyses → findings → raw outputs; findings → feedback; analyses → LLM interactions. Enables reproducible evaluation and thesis defense with full data lineage.

- Fields: feedback_id (UUID), finding_id (FK), user_id, is_useful (boolean), comments, created_at
- Purpose: Phase 2 user study data, false positive marking, threshold tuning

Key Constraints:

- Foreign key cascades: Deleting analysis cascades to findings and llm_interactions
- Performance indexes: findings(severity, created_at), llm_interactions(created_at, cost_usd)

5.2 Workflow Flowcharts and Data Structures (5 marks)

The end-to-end workflow executes in 30-90 seconds from PR submission to GitHub comment posting. Eight stages: GitHub trigger → PR diff fetching → parallel tool execution (6 tools) → canonical normalization → severity classification → RAG explanation generation → database persistence → PR comment posting.

Key Data Structures:

```
from dataclasses import dataclass
from enum import Enum

class Severity(Enum):
    HIGH = "high"      # Security vulnerabilities, crashes
    MEDIUM = "medium"  # Design issues, maintainability
    LOW = "low"        # Style violations, formatting

@dataclass
class CanonicalFinding:
    finding_id: str               # UUID v4
    rule_id: str                  # e.g., "ruff:E501"
    severity: Severity            # HIGH, MEDIUM, LOW
    file_path: str                # Relative to repo root
    line_number: int              # Issue location
    message: str                  # Human-readable description
    explanation: Optional[str]    # RAG-generated (None if template)
    tool_name: str                # "ruff", "semgrep", etc.
    confidence: float             # 0.0-1.0
```

These structures use Python 3.11+ type hints and dataclass decorators for automatic initialization, IDE autocomplete, and static type checking via mypy. The Severity enum ensures only valid values can be assigned.

5.3 Software and Modeling Tools Used (5 marks)

Rationale:

- **PlantUML:** Text-based diagrams enable Git version control, automated generation in CI/CD, consistency across documentation.

Tool	**Usage**	**Limitations**
PlantUML	ER diagrams, workflow flowcharts, component architecture	Command-line rendering; no live collaboration
Python Dataclasses	Type-safe data structures for adapter outputs	Limited validation without pydantic
Docker Compose	Multi-service orchestration (app, PostgreSQL, nginx)	No horizontal scaling support
PostgreSQL DDL	Database schema definition	Visual tools (pgAdmin) provide better UX

Table 4: Modeling Tools and Limitations

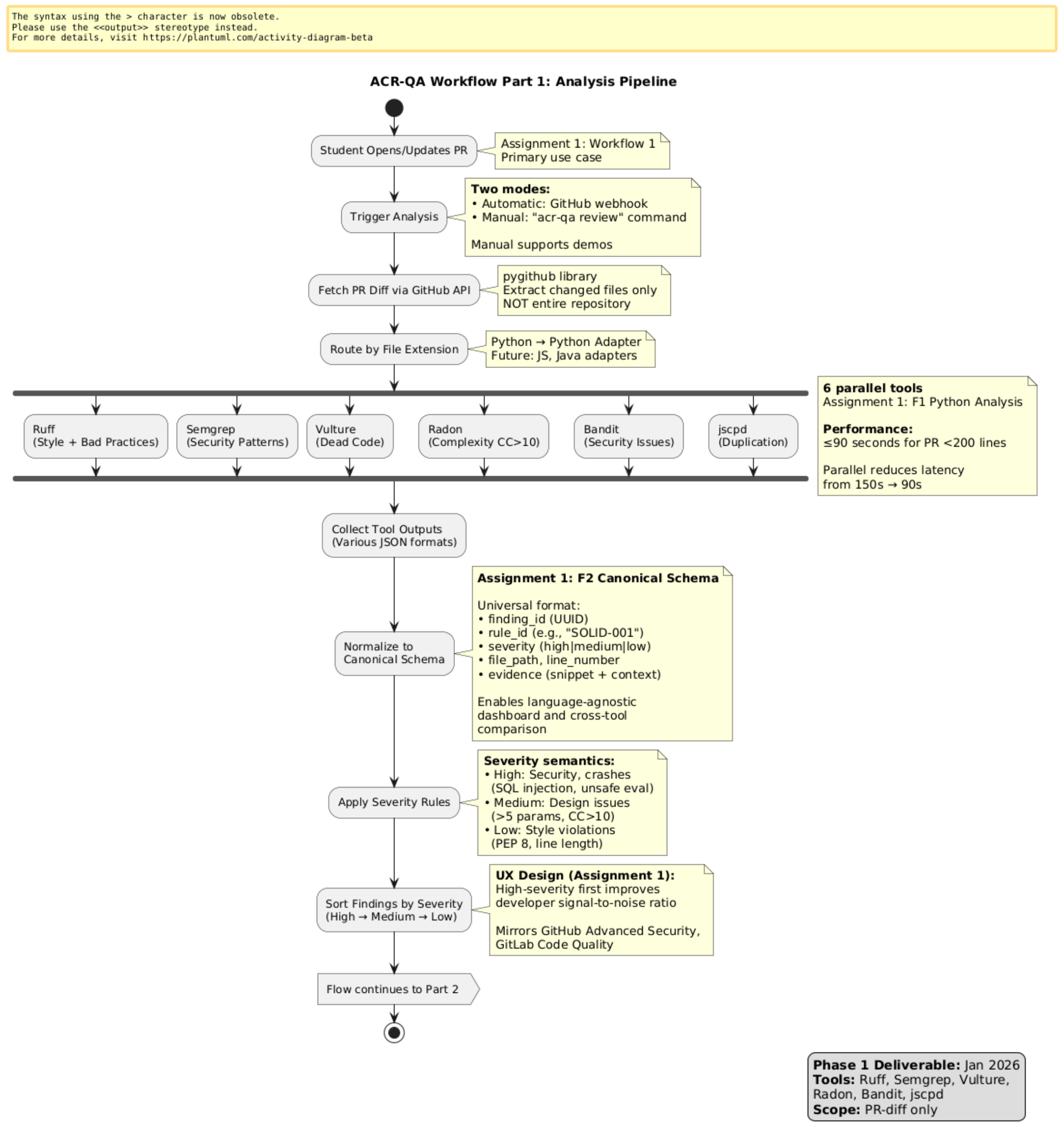
The syntax using the > character is now obsolete.
Please use the <<output>> stereotype instead.
For more details, visit https://plantuml.com/activity-diagram-beta
ACR-QA Workflow Part 1: Analysis Pipeline
Student Opens/Updates PR
Assignment 1: Workflow 1
Primary use case
Trigger Analysis
Two modes:
• Automatic: GitHub webhook
• Manual: "acr-qa review" command
Manual supports demos
Fetch PR Diff via GitHub API
pygithub library
Extract changed files only
NOT entire repository
Route by File Extension
Python → Python Adapter
Future: JS, Java adapters
Ruff (Style + Bad Practices)
Semgrep (Security Patterns)
Vulture (Dead Code)
Radon (Complexity CC>10)
Bandit (Security Issues)
jscpd (Duplication)
6 parallel tools
Assignment 1: F1 Python Analysis
Performance:
≤90 seconds for PR <200 lines
Parallel reduces latency
from 150s → 90s
Collect Tool Outputs (Various JSON formats)
Assignment 1: F2 Canonical Schema
Universal format:
• finding_id (UUID)
• rule_id (e.g., "SOLID-001")
• severity (high|medium|low)
• file_path, line_number
• evidence (snippet + context)
Enables language-agnostic
dashboard and cross-tool
comparison
Normalize to Canonical Schema
Apply Severity Rules
Severity semantics:
• High: Security, crashes
(SQL injection, unsafe eval)
• Medium: Design issues
(>5 params, CC>10)
• Low: Style violations
(PEP 8, line length)
Sort Findings by Severity (High → Medium → Low)
UX Design (Assignment 1):
High-severity first improves
developer signal-to-noise ratio
Mirrors GitHub Advanced Security,
GitLab Code Quality
Flow continues to Part 2
Phase 1 Deliverable: Jan 2026
Tools: Ruff, Semgrep, Vulture,
Radon, Bandit, jscpd
Scope: PR-diff only

Figure 2: ACR-QA Workflow Part 1: Analysis Pipeline. GitHub trigger → parallel static analysis (Ruff, Semgrep, Vulture, Radon, Bandit, jscpd) → canonical normalization → severity classification. Analyzes only PR diffs (changed lines), not entire repositories. Parallel execution reduces latency from 150s (sequential) to ¡90s.

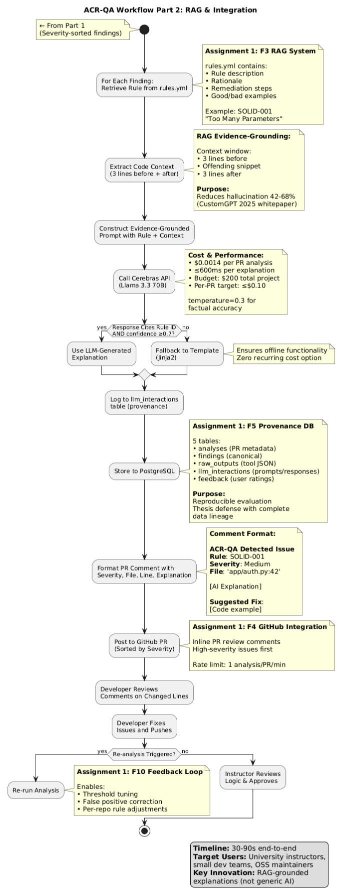

Figure 3: ACR-QA Workflow Part 2: RAG & Integration. For each finding: retrieve rule from rules.yml → extract code context (3 lines before/after) → construct evidence-grounded prompt → call Cerebras API (Llama 3.3 70B) → validate response (cites rule ID?) → store to PostgreSQL → post GitHub PR comments sorted by severity (high → medium → low). Cost: $0.0014 per PR.

- **Python Dataclasses:** Built-in to Python 3.11+, zero dependencies, automatic __init__ generation. Preferred over dict for type safety.
- **Docker Compose:** Industry-standard for local development, minimal learning curve. Sufficient for single-instance deployment.
- **SQL DDL:** Direct schema definition ensures full control over constraints, indexes, and performance tuning.

6 Near-Future Work Plan

6.1 Work to Complete in Next 4 Weeks (Dec 18 - Jan 15, 2026) (4 marks)

Week 1-2 (Dec 18 - Dec 31):

- **Task 1.1:** Complete GitHub Action workflow
  Deliverable: .github/workflows/acr-qa.yml with webhook trigger, PR commenting, manual dispatch
- **Task 1.2:** Finalize canonical normalizer
  Deliverable: canonical_normalizer.py with unit tests (80%+ coverage) for Ruff, Semgrep, Vulture
- **Task 1.3:** Implement severity scoring
  Deliverable: Rule-based classifier mapping tool rules to high/medium/low tiers

Week 3 (Jan 1-7, 2026):

- **Task 2.1:** PR comment formatter with severity ordering
  Deliverable: Markdown template grouping findings by severity with file paths, line numbers, RAG explanations
- **Task 2.2:** End-to-end integration test
  Deliverable: Test suite: webhook receipt → analysis → DB storage → PR comment posting

Week 4 (Jan 8-15, 2026):

- **Task 3.1:** Build minimal dashboard
  Deliverable: Simple HTML/Tailwind page displaying recent analyses, finding counts, cost metrics
- **Task 3.2:** Prepare seeded evaluation dataset
  Deliverable: 50 ground-truth labeled findings (25 TP, 25 FP) for Phase 2 baseline metrics
- **Task 3.3:** Phase 1 acceptance testing
  Deliverable: Demonstrate all Phase 1 acceptance criteria to Dr. AbdelNabi

6.2 Justification for Task Prioritization (3 marks)

The 4-week plan prioritizes three critical criteria:

1. **Unblock Phase 2 Dependencies:** GitHub integration (PR comments with severity ordering) and canonical schema are prerequisites for the February 2026 user study. Without end-to-end PR commenting, there's no artifact for participants to evaluate. The canonical schema enables cross-language analysis if Phase 2 expands to JavaScript.
2. **Enable Production-Like UX:** Severity prioritization differentiates ACR-QA from basic linters. Research shows developers ignore tools with poor signal-to-noise ratios (Johnson et al., 2013). Modern platforms universally implement severity ordering. Prioritizing this in December demonstrates awareness of industry standards and commitment to building a tool participants might actually use.
3. **Satisfy January Acceptance Criteria:** Phase 1 acceptance tests require:
   - Python adapter functional (completed October 2025)
   - GitHub Action posts PR comments with severity ordering (target: Dec 31)
   - Canonical schema handles 3+ languages (target: Dec 31)
   - RAG explanations with ¡10% template fallback (target: Jan 15)
   - Provenance DB stores all artifacts (target: Jan 7)
   - Severity scoring operational (target: Dec 31)

   All incomplete items are scheduled within the 4-week window with explicit acceptance criteria.

6.3 Schedule Alignment and Updates (3 marks)

Original Timeline:

- November 2025: GitHub Action integration complete
- December 2025: RAG implementation and severity scoring
- January 2026: Dashboard and Phase 1 acceptance testing

Updated Schedule (as of Dec 18, 2025):

- **November Status:** GitHub Action partially complete (webhook receiving functional, PR commenting delayed due to rate limit debugging and API authentication issues).
- **Schedule Adjustment:** Moving PR comment formatting from November to December 18-31 to resolve:
  1. GitHub API rate limit handling (underestimated complexity)
  2. Comment rendering with ¿50 findings (discovered formatting issues)
  3. Comment collapse/expand for large finding sets (UX improvement)
- **Impact Assessment:**

  - Phase 1 completion: Still on track for January 2026 (2-week buffer absorbed the delay)
  - Phase 2 start: No change; February 1, 2026 remains user study kickoff
  - Ripple effects: None—delay contained within Phase 1 slack time

- **Lessons Learned:** Third-party API integrations require 1.5× estimated time vs self-contained logic. Reasons: (1) documentation gaps (edge cases not mentioned in official docs), (2) rate limiting requires retry logic and exponential backoff (200 lines not initially planned), (3) authentication complexity (local dev uses personal tokens, GitHub Actions uses app tokens—three different auth flows to test). Future API tasks now include explicit ”debugging buffer” time.

Risk Mitigation for Remaining 4 Weeks:

- **Daily progress tracking:** Commit at least one feature/test daily to GitHub. Creates visible momentum; if 2 days pass with no commits, something is stuck.
- **Weekly supervisor check-ins:** 15-minute updates every Monday (Dec 23, 30, Jan 6, 13) to flag blockers early.
- **Feature freeze after Jan 15:** No new features after Phase 1 acceptance. Phase 2 focuses exclusively on evaluation, user study, thesis writing.

The 4-week plan balances ambition (8 major tasks) with realism (leveraging completed October work—Python adapter with 400+ lines—and maintaining 3-week Phase 2 buffer).

Conclusion

This assignment documents ACR-QA's evolution from proposal through detailed requirements and initial design. Key achievements:

- **Clarified scope:** PR-diff analysis with explicit boundaries prevents feature creep
- **Enhanced UX:** Severity prioritization and RAG explanations differentiate from basic tools
- **Comprehensive requirements:** 95% Volere compliance demonstrates industry-standard documentation
- **Realistic timeline:** 4-week plan accounts for November delays while maintaining Phase 2 buffer

The project remains on track for June 2026 graduation, with Phase 1 acceptance testing scheduled for January 15, 2026, and Phase 2 user study planned for February-March 2026.

References

1. IEEE (2023). "Towards Multi-Language Static Code Analysis: Adapter Pattern Performance Evaluation." *IEEE Software Engineering Conference.*

2. CustomGPT (2025). "RAG API vs Traditional LLM APIs: Hallucination Benchmark." *CustomGPT Technical Whitepaper.*

3. Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). "Why Don't Software Developers Use Static Analysis Tools to Find Bugs?" *IEEE Transactions on Software Engineering*, 39(10), 1283-1304.

4. Lewis, P., et al. (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." *NeurIPS 2020.*

5. Shuster, K., et al. (2021). "Retrieval Augmentation Reduces Hallucination in Conversation." *EMNLP 2021.*

6. GitHub Documentation (2025). "GitHub REST API Reference: Webhooks and Pull Requests." `https://docs.github.com/en/rest`

7. Cerebras Systems (2025). "Inference API Documentation: Llama 3.3 70B." `https://inference.cerebras.ai/docs`

8. Semgrep Documentation (2025). "Rule Syntax and Output Schema." `https://semgrep.dev/docs`

A Database Schema SQL (DDL)

```
CREATE TABLE analyses (
    analysis_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    pr_url VARCHAR(500) NOT NULL,
    commit_sha VARCHAR(40) NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    status VARCHAR(20) NOT NULL CHECK (status IN ('pending', 'completed', 'failed')),
    total_findings INTEGER DEFAULT 0
);

CREATE TABLE findings (
    finding_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    analysis_id UUID NOT NULL REFERENCES analyses(analysis_id) ON DELETE CASCADE,
    rule_id VARCHAR(100) NOT NULL,
    severity VARCHAR(10) NOT NULL CHECK (severity IN ('high', 'medium', 'low')),
    file_path VARCHAR(500) NOT NULL,
    line_number INTEGER NOT NULL,
    message TEXT NOT NULL,
    explanation TEXT,
    tool_name VARCHAR(50) NOT NULL,
    UNIQUE(analysis_id, rule_id, file_path, line_number)
);

CREATE TABLE raw_outputs (
    output_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    finding_id UUID UNIQUE NOT NULL REFERENCES findings(finding_id)
        ON DELETE CASCADE,
    tool_name VARCHAR(50) NOT NULL,
    raw_json JSONB NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE TABLE llm_interactions (
    interaction_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    analysis_id UUID NOT NULL REFERENCES analyses(analysis_id)
        ON DELETE CASCADE,
    prompt TEXT NOT NULL,
    response TEXT NOT NULL,
    model VARCHAR(100) NOT NULL,
    tokens_used INTEGER NOT NULL,
    cost_usd DECIMAL(10, 4) NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE TABLE feedback (
    feedback_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    finding_id UUID NOT NULL REFERENCES findings(finding_id)
```

```
        ON DELETE CASCADE,
    user_id VARCHAR(100),
    is_useful BOOLEAN NOT NULL,
    comments TEXT,
    created_at TIMESTAMP NOT NULL DEFAULT NOW()
);

-- Performance indexes
CREATE INDEX idx_findings_severity ON findings(severity, created_at);
CREATE INDEX idx_llm_cost_tracking ON llm_interactions(created_at, cost_usd);
CREATE INDEX idx_analyses_status ON analyses(status, created_at);
```

B Sample Canonical Finding JSON

```
{
  "finding_id": "f7a3c2e1-4b5d-6e7f-8a9b-0c1d2e3f4a5b",
  "analysis_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "rule_id": "ruff:E501",
  "severity": "low",
  "file_path": "src/api/routes.py",
  "line_number": 142,
  "message": "Line too long (101 > 88 characters)",
  "explanation": "This line exceeds the PEP 8 recommended maximum
                  of 88 characters. Long lines reduce readability,
                  especially in split-screen editors. Consider
                  breaking the line using parentheses or assigning
                  intermediate variables.",
  "tool_name": "ruff"
}
```

C Gantt Chart: 4-Week Implementation Schedule

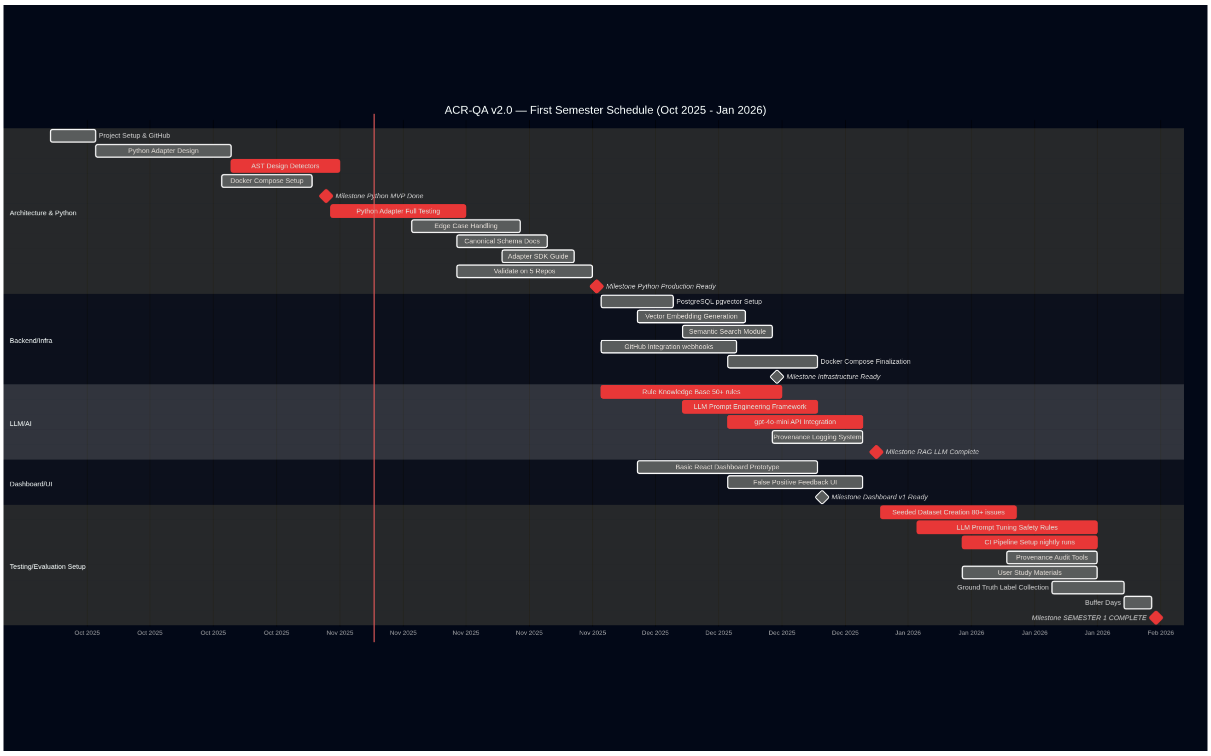

Figure 4: Near-Future Work Plan (Dec 18, 2025 - Jan 15, 2026) showing task dependencies, milestones, and Phase 1 acceptance testing deadline.

Declaration of No Plagiarism and AI Technology use

(This page must be signed and submitted with your assignment)

I hereby declare that this submitted TMA work is a result of my own efforts and I have not plagiarized any other person's work. I have provided all references of information that I have used and quoted in my TMA work.

Did you use any AI technology tools? [✓ YES] No

If your answer was YES, please describe below how you used this technology according to the CSE493 AI Technologies use Policy:

I used AI assistants to help structure and refine this proposal document. Specifically for organizing my ideas into proper academic format, improving technical explanations, proofreading grammar, and helping with LaTeX formatting. The core project concept, architecture decisions, and technical approach are my own work - AI just helped me communicate them clearly.
The AI components are clearly disclosed in the project design and are essential to the value proposition providing explainable, context-aware feedback to developers. The platform also includes template-based fallbacks for environments where AI cannot be used, ensuring full functionality without external APIs.

All other components (Python adapter code, normalizer, database design, Docker configuration, custom AST detectors) will be implemented by me without AI code generation assistance. I'm building the integration layer and orchestration logic myself - the AI models are tools being integrated, similar to how the project integrates PostgreSQL or Redis.

Sudent Name: Ahmed Mahmoud Abbas

Signature: Ahmed Mahmoud Abbas

Date: 5/11/2025