

# Assignment 1 - Detailed Feedback Report

Student ID: 38961

## Assessment Overview

Your submission demonstrates a very solid foundation in software engineering practices and LLM application development. You have successfully implemented most of the advanced requirements with only minor areas for enhancement. Your work shows strong attention to detail, comprehensive documentation, and professional development practices.

## Areas of Excellence

Your submission demonstrates particular strength in the following areas:

### Ui Ux

- Found 36 image(s)/screenshot(s)
- Excellent visual documentation with 5+ images
- README includes UI screenshots/documentation

### Project Planning

- PRD.md found
- Functional Requirements found in PRD
- ARCHITECTURE.md found

### Testing Quality

- Found 8 test file(s)
- Excellent test coverage with 8 test files

- Test framework configured: tests\pytest.ini

## Research Analysis

- Found 1 Jupyter notebook(s) and 8 research document(s)
- Multiple research artifacts for comprehensive analysis
- Visualizations found in 1 notebook(s)

## Version Management

- Good commit history: 12 commits
- Excellent commit message quality (100% meaningful)
- Prompt documentation found: documentation\Prompting\_and\_Developing.md

## Config Security

- No hardcoded secrets found in production code
- .env.example file found
- .gitignore file found with .env properly ignored

## Code Documentation

- README.md found (54051 bytes)
- Installation instructions found in README
- Usage examples found in README

## Areas for Improvement and Development

While your submission is strong overall, the following areas present opportunities for minor enhancements that would further strengthen your work:

## Quality Standards

Current Status: Code quality information in README: README.md (quality standards section)

Recommended Actions:

- Configure linting tools (pylint, eslint, etc.)
- Set up CI/CD pipeline (GitHub Actions, GitLab CI, etc.)
- Create dedicated style guide or CONTRIBUTING.md
- Set up pre-commit hooks for automatic quality checks

## Priority Action Items

### High Priority Enhancements:

These areas would significantly strengthen your submission and should be addressed after the immediate focus items:

- Improve Quality Standards

## Conclusion

Your submission demonstrates excellence in software engineering and LLM application development. The minor enhancements suggested above will help you maintain and further develop your already strong skills. Continue to build on this solid foundation, and focus on the advanced aspects that distinguish exceptional work from good work.

Assessment Date: 2025-12-04

This report provides developmental feedback to support your learning and growth.

# **Assignment 1 - LLMs in Multi-Agent Environment**

---

**Group Code:** LLM\_Agents\_Tom\_Igor\_Roie

**Submitters:**

Tom Ron, ID 301020723

Igor Nazarenko, ID 322029158

Roie Gilad, ID 312169543

**GitHub Repository:**

[https://github.com/tomron87/LLM\\_Agent\\_Orchestration\\_HW1](https://github.com/tomron87/LLM_Agent_Orchestration_HW1)

**Self-Assigned Grade: 92/100**

# Self-Evaluation Justification Form

---

## Students:

Tom Ron, ID 301020723

Igor Nazarenko, ID 322029158

Roie Gilad, ID 312169543

**Project Title:** Assignment 1 - LLMs in Multi-Agent Environment

**Submission Date:** November 2025

**Self-Assigned Grade:** 92/100

---

## Justification for Our Self-Evaluation

We believe our project merits a 92/100 grade based on comprehensive evaluation across all required criteria. Our primary strengths include production-grade three-layer architecture with 89% test coverage across 35 tests, comprehensive documentation suite of 10 files including detailed PRD with KPIs, C4 architecture diagrams at 4 levels, and 7 industry-standard ADRs. We demonstrated academic rigor through systematic parameter sensitivity analysis with Jupyter notebook visualizations. Our security implementation is exemplary with zero hardcoded secrets, comprehensive .gitignore (359 lines), and entropy-enforced preflight validation.

However, we acknowledge honest weaknesses preventing a perfect score. Our prompt engineering documentation in `Prompting_and_Developing.md` lacks depth and detailed examples—we inadequately captured actual prompts and iterative refinement processes. Initially, our weak prompt engineering skills caused substantial time loss as we struggled with Claude and ChatGPT, requiring frustrating iterations before achieving results. Additionally, our research documentation lacks academic citations (3-5 needed), and the Jupyter notebook's LaTeX formulas need expansion. These represent genuine improvement areas identified through self-reflection.

We invested approximately 70 hours on this project, distributed across architecture design, implementation, testing, and documentation. The time investment reflects not just coding hours but a steep learning curve—particularly in mastering AI-assisted development workflows and understanding production-grade patterns. Significant time was spent on iterative refinement: experimenting with three-layer architecture decisions, achieving comprehensive test coverage with proper unit/integration separation, and creating documentation that genuinely serves future developers rather than checking boxes. While the hour count may seem modest, the density of learning and quality-over-speed approach resulted in production-grade deliverables.

Our genuine innovations go beyond the basic Ollama integration requirement. The `Helper Module Pattern` (`ui/components.py`) solves Streamlit's testability problem by separating visual rendering from business logic. Our documentation-as-code approach with `CLAUDE.md` creates a living development guide enabling effective AI-agent collaboration. The sophisticated test architecture with `DummyResp` fixtures enables true unit testing without mocking libraries—a pattern we developed through experimentation. The notice field pattern returns helpful Hebrew guidance within successful responses rather than throwing errors, prioritizing user experience. Finally, our fail-fast configuration validation and Makefile-driven workflow transform a student project into a system you could hand to a junior developer without explanation. These patterns represent architectural thinking, not just feature completion.

The most valuable outcome was our transformation in AI-assisted development. We progressed from vague requests to precise, mission-oriented prompts with clear success criteria. We mastered breaking complex tasks into structured missions, planning architecture through AI conversations, and validating outputs systematically. Designing three-layer architecture through AI-guided iterations taught us both software engineering principles and leveraging AI as a force multiplier, making this a transformative learning experience in next-generation development practices.

