Assignment 3 - Multi-Agent Translation Analysis

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/roiegilad8/LLM_Agent_Orchestration_HW3

Self-Assigned Grade: 95/100

November 25, 2025

1 Project Overview

This project implements a multi-agent translation analysis system measuring semantic drift across a three-stage pipeline: English → French → Hebrew → English. The system uses Ollama local LLMs (llama3.2:3b) with controlled error injection to study the relationship between spelling errors and semantic preservation.

1.1 Key Deliverables

- **Architecture:** Multi-agent orchestration with BaseAgent abstraction and AgentChain orchestrator
- **Testing:** 43 comprehensive tests with 81% code coverage, including mocking fixtures in `conftest.py`
- **Documentation:** PRD, 3 ADRs (architecture, embedding model, error injection), README with installation guide, and analysis Jupyter notebook
- **CI/CD:** GitHub Actions workflow for automated testing with 80% coverage threshold
- **Cost Efficiency:** $0.00 operational cost using local Ollama models

1.2 Technical Implementation

The system uses Sentence-BERT embeddings to calculate semantic distance via cosine similarity. Experiments tested six error rates (0-50%) across three sentences with three repetitions each, revealing a linear relationship between error injection and semantic drift. The CLI provides three commands: single translation, full experiment execution, and results analysis with visualizations.

2 Self-Assessment Justification

2.1 Grade: 95/100

2.1.1 Strengths - What We Did Exceptionally Well

Our project demonstrates several areas of excellence that justify this high grade:

Professional Architecture & Code Quality: We implemented a clean multi-agent design using the BaseAgent abstract class, enabling extensibility and testability. The dependency injection pattern allows seamless mocking of the Ollama API, reducing test execution from 20 minutes to 30 seconds. Type hints throughout the codebase, comprehensive error handling, and a well-structured CLI using Typer demonstrate production-level code quality.

Comprehensive Testing: With 43 tests achieving 81% coverage (exceeding the 70% requirement), we implemented professional testing practices. The `conftest.py` file contains pytest fixtures for mocking Ollama responses, making tests deterministic and CI/CD-ready. Our test suite covers edge cases including empty strings, zero/maximum error rates, reproducibility with seeds, and the new CostTracker utility.

Thorough Documentation: We produced complete documentation including a detailed PRD with KPIs and acceptance criteria, three comprehensive ADRs documenting key architectural decisions, a professional README with installation instructions (including Ollama setup for Linux/macOS/Windows), cost breakdown table, and an analysis notebook with statistical insights and visualizations.

DevOps & Best Practices: Our GitHub Actions CI/CD pipeline automates testing on every push, enforcing the 80% coverage threshold. The CostTracker utility monitors API usage and runtime, demonstrating professional resource awareness even for free services. We maintained reproducible experiments with seed control and locked dependencies in requirements.txt.

2.1.2 Weaknesses - Areas for Improvement

Despite our strong performance, we acknowledge several areas that prevented a perfect score:

Git Commit Quality (-3 points): Some early commits used generic messages like "Add files via upload" rather than descriptive conventional commit messages. While our later commits improved significantly (e.g., "Docs: Add Ollama installation instructions"), consistency from the start would have been better.

Minor Documentation Gaps (-1 point): One ADR has minor formatting inconsistencies, and we could have included more detailed troubleshooting scenarios in the README.

Optimization Opportunities (-1 point): While our mocking reduced test time dramatically, we could further optimize by parallelizing independent test execution or implementing caching for embedding computations.

2.1.3 Time Investment & Effort

This project required approximately **40-50 hours** of collaborative work:

- Architecture design & implementation: 15 hours

- Testing development & mocking: 10 hours
- Experimentation & analysis: 8 hours
- Documentation (PRD, ADRs, README, notebook): 10 hours
- CI/CD setup & debugging: 4 hours
- Code reviews & refinement: 5 hours

The team worked collaboratively, with pair programming sessions for complex components like the AgentChain orchestrator and regular code reviews ensuring quality.

2.1.4 Innovation & Unique Aspects

Our project demonstrates several innovative elements:

Zero-Cost Architecture: Unlike typical LLM projects relying on expensive APIs, we achieved full functionality with $0.00 operational cost using Ollama, making the system sustainable for research and education.

Professional Mocking Framework: Our `conftest.py` implementation showcases industry-standard testing practices rarely seen in academic projects, with fixtures that completely eliminate external dependencies.

Cost Tracking System: The CostTracker utility, while reporting $0 for free services, demonstrates professional resource awareness and would seamlessly adapt if we switched to paid APIs.

Comprehensive CI/CD: The GitHub Actions workflow with coverage enforcement ensures code quality automatically, a best practice often skipped in academic work.

2.1.5 Learning Outcomes

This project provided valuable insights:

Multi-Agent Orchestration: We learned to design clean agent interfaces using abstract base classes, manage sequential workflows with proper error propagation, and inject dependencies for testability.

LLM Integration Best Practices: Working with Ollama taught us local LLM deployment, prompt engineering for translation tasks, and handling multilingual text (English, French, Hebrew) with proper UTF-8 encoding.

Testing Philosophy: We discovered that comprehensive mocking isn't just about speed—it's about reliability, reproducibility, and enabling continuous integration. Our tests now run anywhere, anytime, with identical results.

Documentation as Communication: Writing ADRs forced us to articulate our design decisions clearly, which improved team alignment and made onboarding theoretical new developers trivial.

Semantic Similarity Metrics: Understanding cosine distance versus Euclidean distance, choosing appropriate embedding models (Sentence-BERT for multilingual support), and interpreting semantic drift measurements deepened our NLP knowledge.

The most important lesson was that **quality requires upfront investment in architecture and testing**, which pays dividends in debugging time, confidence in results, and professional presentation.

Academic Integrity Declaration

We, the undersigned, declare that this submission is our own original work. We have:

- Completed this assignment independently as a group
- Properly cited all external sources, libraries, and tools used
- Not copied code from other students or unauthorized sources
- Not shared our code with other students
- Followed all academic integrity guidelines of the course

We understand that violations of academic integrity may result in penalties including failure of the assignment or course.

Signatures:

Tom Ron, ID 301020723	Date: November 25, 2025
Igor Nazarenko, ID 322029158	Date: November 25, 2025
Roie Gilad, ID 312169543	Date: November 25, 2025