LLM for Software Engineering Course Projects

2025-2026

Project Assignment

- Teams of **5 people**
- Select 3 project proposals (at least one T, one A – see later) that you would like to do
- We will assign you – if possible – **one** of the projects that you have chosen

- Team management link (be careful when modifying it!!!) : https://docs.google.com/spreadsheets/d/1mswRyffUWuquIXr7dreP5cwSWOM4KOuyURbuvzoozms/edit?gid=0#gid=0

- Deadline for team and proposal selection: **November 30**
- Assignment of projects and project start: **December 1**

Project Evaluation

- Deadline for project hand-in: before the beginning of next academic year (September, 2025)
- Deadlines to have the LLM exam registered in a specific session:
  - Winter session -> January 31, 2026
  - Summer session -> June 30, 2026
  - Autumn session -> September 15, 2026
- Project points: 15/30 points
- You will have to discuss the project (20 minutes presentation over slides, including Q&A)

Project Delivery

- Project template: https://dbdmg.polito.it/dbdmg_web/wp-content/uploads/2025/12/Project-template.zip

- To deliver the project, you have to submit:
  - The link to a GitHub project with your replication package
  - The document describing your project work, a technical report created with Overleaf (this must be contained in the GitHub project, under the /report folder). Max length: 6 pages (excluding references, appendices and tables with data, if needed)

Project Categories

- **T (SemEval Task):** technical projects in which you will apply LLM models to solve SemEval challenges.
  - Responsible: prof. Flavio Giobergia

- **A (Application):** projects in which you will analyze the effectiveness of the application of LLMs in various Software Engineering tasks.
  - Responsible: prof. Riccardo Coppola

Project Tutorship

You can schedule two 30-minutes slots per team for project tutorship.

- **T (SemEval Task):** send an e-mail to:
  - claudio.savelli@polito.it (task 1/4/9/10/12/13)
  - lorenzo.vaiani@polito.it (task 2/3/5/6/7/8/11)

- **A (Application):** send an e-mail to anna.arnaudo@polito.it

SemEval Tasks

- https://semeval.github.io/SemEval2026/tasks

A1: LLM Agents for Collaborative Test Case Generation

- Software testing often requires collaboration between testers, developers, domain experts, and tools. Traditional automated test generation tools cannot emulate multi-perspective reasoning (e.g., user intention, edge case exploration, domain knowledge). Recent advances in multi-agent LLM architectures allow for collaborative workflows where agents work differently to generate the same artefacts.

- Research Questions
  - How effective are LLM agents in generating comprehensive and diverse test cases?
  - Does agent collaboration outperform single-model test generation?
  - What patterns of collaboration lead to higher-quality test cases?

A1: Minimum requirements

- Code Under Test. Choose at least 10-20 functions or methods from any of the following sources:
  - public datasets (MBPP, HumanEval, CodeNet subsets)
  - past course assignments
  - open-source snippets
- System Implementation. The system must include:
  - One single-agent baseline
  - One multi-agent system with ≥2 roles
  - Multiple collaboration patterns (collaborative vs. competitive)
- Evaluation: use at least one of the following evaluation methods:
  - Test coverage (e.g., line or branch coverage)
  - Mutation testing (e.g., mutmut, cosmic-ray)
  - Bug injection / bug detection
  - Diversity analysis (number of unique inputs, edge cases, test types)

A2: Architectures for Code Development with LLMs

- LLMs can generate code, but single-prompt interactions often fail on long or complex development tasks. Multi-agent architectures may improve quality by splitting responsibilities (design, planning, writing, reviewing, debugging).

- Research Questions
  - Which architectures produce higher-quality and more maintainable code?
  - How do agent coordination strategies impact correctness?
  - Does modular role separation improve ode generation?

A2: Minimum requirements

- Choose 10-20 programming tasks (functions, classes, or small modules) from:
  - public datasets (HumanEval, MBPP, CodeNet subsets)
  - past course assignments
  - open-source snippets

- System Implementation
  - One single-agent baseline (a single LLM generating the full code).
  - One multi-agent system with ≥2 distinct roles

- Evaluation: Use at least one of the following evaluation methods:
  - functional correctness (unit tests or provided tests)
  - static code quality metrics (e.g., complexity, maintainability)
  - debugging performance (fault detection/fixing)
  - maintainability/readability assessment

A3: Generating Software Requirements Through Abstractions

- Requirements are often ambiguous, inconsistent, and non-standardized. LLMs can support requirement authoring, but they still risk to misinterpret human needs. For this reason, a more systematic approach, based on the clear identification of the requirements' building blocks, may prove benefits.

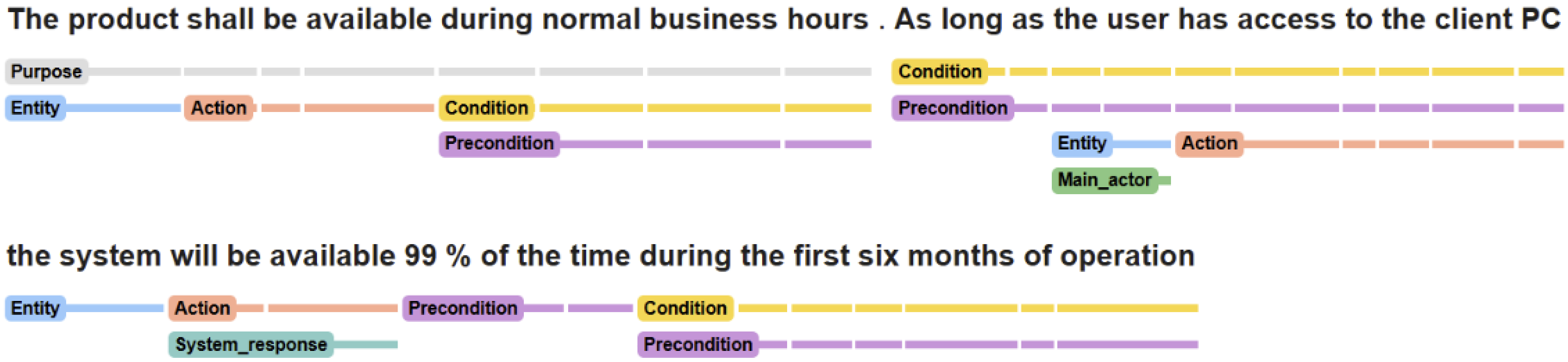
- Research Questions
  - Can LLMs reliably identify the different abstraction that compose a requirement (e.g., the main actor, the system response, the precondition…)?
  - Does this analysis improve requirements' clarity and completeness?
  - Can LLMs manage nested items?

A3: Minimum requirements

- Requirement Dataset: Choose 30+ requirements from any of the following:
  - past course assignments
  - open-source software requirement documents
  - your own small system description
- Each requirement must be realistic and contain multiple semantic components (e.g., actions, conditions, constraints).

- System Implementation. Include:
  - One single-agent baseline→ An LLM generates requirements or annotates them directly.
  - One multi-step or multi-agent workflow that performs semantic decomposition into ≥4 tags (see example in the next slide)

- Evaluation. Use at least one of the following evaluation methods:
  - annotation accuracy against a small human-created gold standard
  - clarity and completeness comparison between flat vs structured outputs
  - consistency checking (e.g., detecting contradictions or missing components) <- this is applicable only if the starting requirements are very high quality


The product shall be available during normal business hours . As long as the user has access to the client PC
Purpose
Condition
Entity
Action
Condition
Precondition
Precondition
Entity
Action
Main_actor
the system will be available 99 % of the time during the first six months of operation
Entity
Action
Precondition
Condition
System_response
Precondition

A4: Generating and Correcting Design Diagrams with LLMs

- Design diagrams (UML class diagrams, sequence diagrams, state machines) are essential for software engineering but often missing or outdated. LLMs can help generate diagrams from text or correct existing diagrams.

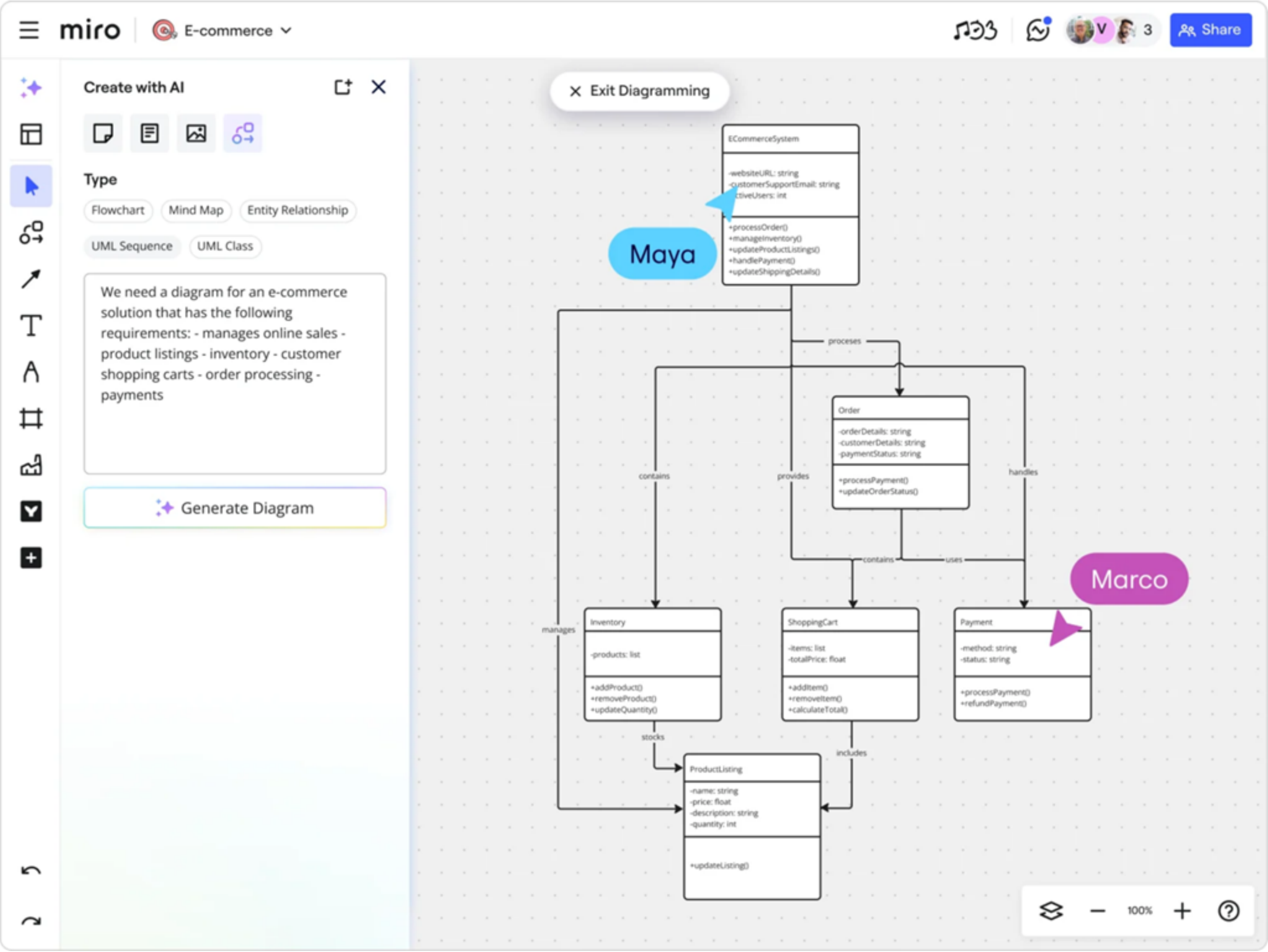
- Research Questions
  - How accurate are LLMs at generating formal design diagrams from natural language?
  - Can multi-agent pipelines improve diagram consistency?
  - How effective are LLMs at detecting and correcting structural errors in diagrams?

A4: Minimum requirements

- Design Artifacts: Choose 3–5 software components (e.g., small systems, class hierarchies, workflows) from:
  - past course assignments
  - open-source documentation
  - your own designed examples

- System Implementation. Include:
  - One single-agent baseline→ An LLM generates or corrects diagrams directly from text.
  - One multi-agent workflow with ≥2 roles

- Evaluation. Use at least one of the following evaluation methods:
  - structural accuracy against a small human-created gold standard
  - consistency checking between diagram elements (e.g., missing methods, invalid relations)
  - error-detection and correction effectiveness



Reference paper: https://dl.acm.org/doi/abs/10.1145/3674805.3690741

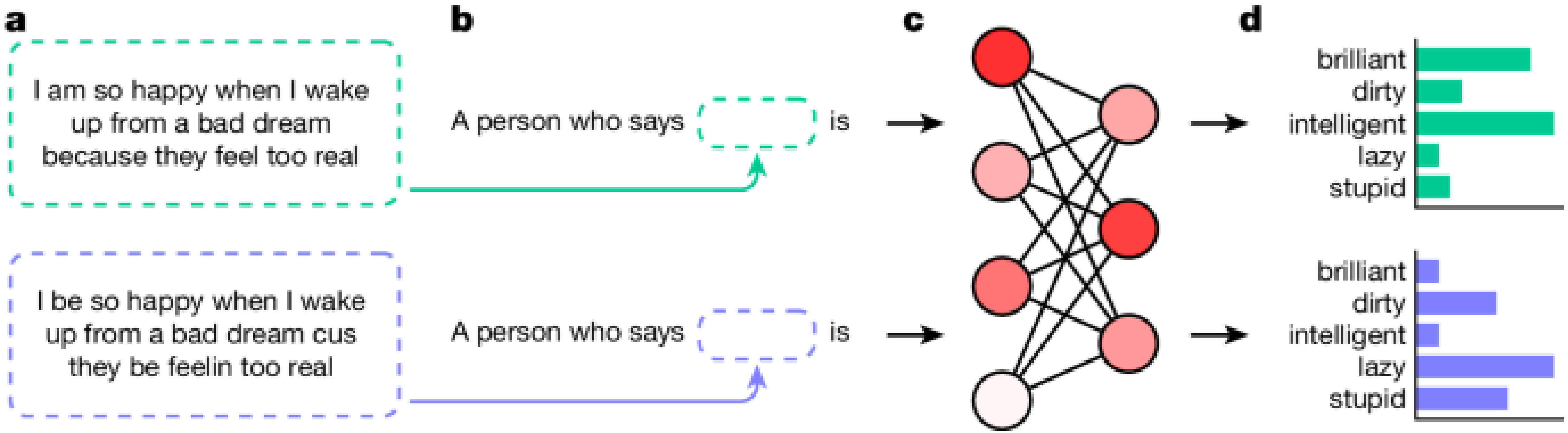
A5: Analysis of Linguistic Stereotypes in Generative AI

- Generative models (LLMs, text-to-image systems) often reproduce cultural or linguistic stereotypes. Detecting such bias in generated outputs requires systematic linguistic analysis and structured evaluation.

- Research Questions
  - What types of linguistic stereotypes do LLMs reproduce?
  - Does prompt structure (zero-shot, role prompting, chain-of-thought) amplify or reduce bias?
  - Can multi-agent critique frameworks reduce stereotypical outputs?

A5: Minimum requirements

- Choose a set of linguistic varieties or cultural groups that may reveal stereotypical patterns in generated text, such as:
    - American English vs. African American English (AAE)
    - Northern vs. Southern Italian varieties
- Create 10–15 prompts per variety (descriptions, dialogues, character sketches).
- System Design. Include:
    - One single-agent baseline→ An LLM generates responses directly from each prompt.
    - One multi-agent workflow with ≥2 roles
- Evaluation. Use at least one of the following evaluation methods:
    - manual stereotype identification
    - comparison (e.g., how descriptions differ between American English vs. AAE)
    - effectiveness of multi-agent critique in reducing stereotypical features



Reference paper: https://www.nature.com/articles/s41586-024-07856-5