# Assignment 2: Self-Reflection via MultiAgent Collaboration

Student ID: 111502026, Name: 薛耀智

## 1. (20%) Multi-Agent System Design for Task Collaboration

How can you design a Multi-Agent system to collaboratively complete tasks in a web-based environment? For example, how can agents coordinate to perform actions such as form filling, submission, and data searching?

When designing a Multi-Agent System (MAS) capable of collaboratively completing tasks in a web environment, the core principle lies in decomposing a task into multiple independently executable subtasks, each assigned to agents equipped with specialized capabilities.

At the center of the system, a Coordinator Agent can be introduced to manage task delegation and track progress based on the user's overall goal. Then, for common web-based operations such as form filling, data retrieval, and submission, dedicated agents can be defined: a Form Agent, a Search Agent, and a Submission Agent.

The Search Agent is responsible for extracting target information from web pages or databases based on specific conditions. The retrieved data is then passed to the Form Agent, which automatically populates relevant form fields. Once the form is completed, the Submission Agent simulates user actions such as clicking submit buttons or calling APIs to finalize the process.

These agents coordinate through a shared task state storage or inter-agent communication channels, such as message queues or event-driven systems, ensuring synchronized task execution while maintaining a clear division of responsibilities.

In case of exceptional conditions — such as failed field validation or no search results — an Error Handling Agent can be integrated to implement fallback procedures or retry mechanisms. This enhances the system's robustness and flexibility in dynamic environments.

Such a modular and interoperable architecture significantly boosts the scalability and adaptability of web task automation, allowing the system to handle increasingly complex workflows across various domains.

## 2. (20%) Error Analysis and Strategy Adjustment in AI Agents

How can an AI agent analyze its own errors after performing web-based operations, and adjust its strategy accordingly?
What techniques or mechanisms can be used to identify failure patterns and improve future performance?

In the movie search–oriented Multi-Agent System (MAS) I designed, AI agents are not only capable of planning and tool execution but also demonstrate advanced abilities in error analysis and strategic adjustment. This capability is enabled through the design of a specialized agent known as the Replanner. When an agent

encounters discrepancies between the search results and the user's input — for example, if the retrieved movie's plot contradicts the original query, or if the listed actors don't match the ones mentioned by the user — the system triggers an error analysis mechanism. The Replanner receives the original task input, the initial plan, all completed steps, and their intermediate outputs. It then performs semantic analysis and reasoning to identify where the task may have gone off track and adjusts the strategy accordingly. This may involve replacing the proposed movie title, modifying search conditions, or selecting an alternative candidate — all aimed at avoiding the repetition of previous errors. Such strategic adaptation relies on the large language model's capabilities in contextual understanding and logical reasoning.

Additionally, the system maintains a structured task state using LangGraph's StateGraph, along with a step-by-step execution history (e.g., a growing past_steps list). This design ensures that every action and response is preserved, enabling the Replanner to detect patterns of failure — such as recurring keyword mismatches or poor tool performance under specific input conditions. This form of pattern recognition and retrospective analysis allows the system to learn from each interaction, resulting in more effective strategy selection over time. Consequently, the MAS continuously improves its success rate and enhances user experience in similar tasks. With future integration of logging and statistical analysis, the system could evolve into one that supports continuous learning and long-term optimization, giving the MAS the potential to grow smarter and more reliable over time.

## 3. (20%) Reflection Strategies in Agentic Systems

What reflection strategies can be used in AI agents or Multi-Agent systems to improve performance?
For example, how can self-reflection, peer-review, or planning revisions be implemented after task execution?

In the movie search–oriented Multi-Agent System (MAS) I designed, performance optimization and strategic adjustment are effectively achieved through the integration of self-reflection, planning revision, and a logic resembling peer review. The core mechanism enabling this is the Replanner Agent, which actively analyzes whether the intermediate or final results align with the original user goal. When inconsistencies arise — such as semantic mismatches, incomplete information, or incorrect responses — the Replanner conducts a reflective review of the entire task history, including the original input, the initial plan, and all previously executed steps. Through this self-reflective reasoning and re-planning, it generates a revised sequence of remaining steps. This process allows the system to inspect whether earlier reasoning and execution strategies were valid and adapt future actions accordingly — a hallmark of reflective learning in AI systems.

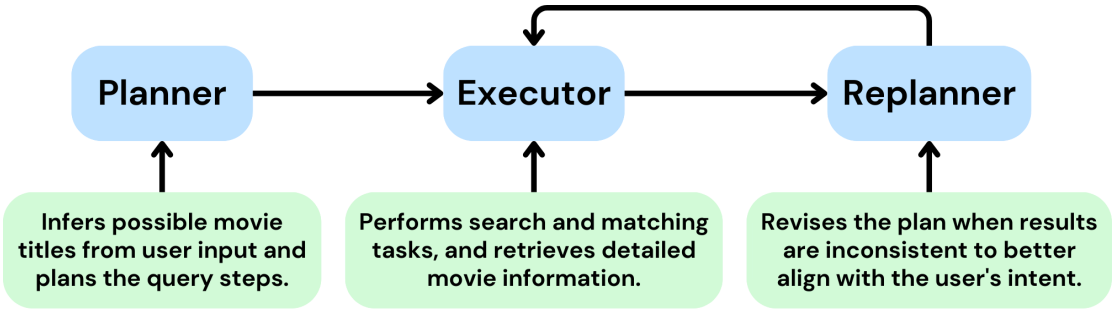
Furthermore, the MAS architecture exhibits a form of peer-review-like logic: the roles of the Planner and Executor are clearly delineated — the Planner devises the strategy, while the Executor carries out the steps. If the results are suboptimal, the Replanner critiques and proposes modifications to the original plan. This dynamic creates a collaborative interaction where agents evaluate one another's contributions and identify potential flaws, thereby enhancing the MAS's overall responsiveness and robustness.

From a technical standpoint, these reflective strategies are implemented using LangGraph's state machine framework and semantic memory tracking mechanisms

— such as the accumulation of past_steps and the assignment of thread IDs — which enable effective state backtracking and context retention. Each round of reflection and replanning is thus grounded in a reliable memory of the task flow. This design not only provides flexibility and stability but also lays the groundwork for future enhancements such as automatic evaluator agents, independent auditing agents, or reinforcement learning components to further advance the system's adaptive intelligence.

## 4. (10%) Agentic AI System Architecture

Describe the architecture of an Agentic AI system with a diagram.
Illustrate and explain the core components—such as Perception, Reasoning (Brain), Action, and Reflection—and how they interact. If applicable, show how multiple agents collaborate within this architecture.




In the Multi-Agent System (MAS) designed for this project, we implemented an Agentic AI architecture that incorporates the key cognitive capabilities of Perception, Reasoning (Brain), Action, and Reflection. The system is composed of three main agents — Planner, Executor, and Replanner — which work collaboratively to complete the end-to-end task of interpreting user input and retrieving accurate movie information. The interaction between these agents is clearly illustrated in the provided diagrams and flowcharts.

The process begins with the Perception phase, where the system receives natural language input from the user. These inputs may contain clues such as plot descriptions, actor names, or release years. The perceived input is semantically parsed and passed on to the Planner for reasoning.

In the Reasoning (Brain) phase, the Planner plays the central reasoning role. It analyzes the user's input, infers one or more possible movie titles, and generates a structured task plan. This plan outlines a sequence of steps — from generating title candidates to searching for matching movies and finally to retrieving detailed information — that guides the execution phase.

During the Action phase, the Executor carries out the steps proposed by the Planner. For instance, based on the generated movie title, it performs an IMDb search, retrieves multiple candidate results, and compares them against the user's description using attributes like cast, plot, and release year. Once the most relevant movie is identified, the Executor invokes specialized tools to extract detailed movie data, including a synopsis, actor list, and release date.

If, at any point, the system detects that the retrieved results do not align with the user's intent, it transitions into the Reflection phase. The Replanner is responsible for reviewing the task history — including the original input, the current plan, and all completed steps — and determining whether the task strategy should be revised.

Adjustments may include proposing a new movie title, changing search parameters, or reselecting candidate results. This reflection process enhances the system's flexibility and introduces self-monitoring and correction capabilities.

The interaction among these three agents forms a closed-loop cycle: Planner → Executor → Replanner → (back to Planner or Executor, if needed). This is not merely a linear task execution pipeline but an adaptive system architecture capable of reflection and re-planning. Within this MAS structure, each agent focuses on a specialized subtask, and their coordination is facilitated through clear task logic and structured information flow. As a result, the system maintains high stability and accuracy even when handling vague or uncertain user inputs. This design concept can be further extended to broader domains, such as task automation, intelligent assistants, and other adaptive AI systems.