Building Agents that Learn Like Humans – “The Second Mind”

Introduction

Create "The Second Mind," a system of AI agents that mimics human learning—retaining preferences, connecting ideas, and improving with each interaction. Your challenge is to implement a coalition of specialized agents that iteratively refine outputs toward a research goal, incorporating real-time web data extraction for enhanced research.

Objective

Build a proof-of-concept that:

1. Stores and recalls past interactions using a basic retrieval and update mechanism.
2. Uses six specialized agents—Generation, Reflection, Ranking, Evolution, Proximity, and Meta-review—managed by a Supervisor agent.
3. Extracts information from the web in real-time to inform research.
4. Shows iterative improvement in scientific reasoning or hypothesis generation.

Problem Description

Design a system where:

- **Specialized Agents** have distinct roles:
  - **Generation**: Creates initial hypotheses.
    Example: Suggests "solar panels on rooftops" for "urban renewable energy."
  - **Reflection**: Checks coherence.
    Example: Flags "solar panels need sunlight" as relevant based on web data.
  - **Ranking**: Scores outputs.
    Example: Rates "solar panels" (8/10) using real-time cost data from the web.
  - **Evolution**: Refines ideas.
    Example: Upgrades to "solar window panels" after web search on urban trends.
  - **Proximity**: Links to past interactions.
    Example: Recalls "solar worked in a past urban query" from memory.
  - **Meta-review**: Evaluates the process.
    Example: Notes "web data slowed Ranking; optimize next time."
- A **Supervisor Agent**:
  - Assigns tasks to agents in a queue.
    Example: Sends "fetch web data" to Generation, then "score" to Ranking.
  - Allocates resources dynamically.
    Example: Prioritizes Evolution after web data reveals new options.
  - Enables feedback loops for improvement.
    Example: Loops refined ideas back to Ranking with updated web insights.
- The system stores and retrieves interactions and extracts real-time web data.
  Example: Stores "solar panels scored 8/10" and fetches "latest solar costs" from a site.

Example Workflow

Input: "Renewable energy for urban areas."

- Cycle 1: Generation pulls "solar panels" from a web article; Reflection confirms relevance; Ranking scores 8/10 using web-sourced prices.
- Cycle 2: Evolution refines to "solar window panels" after a web search on innovations; Proximity links to past solar success; Ranking ups score to 9/10.
- Meta-review suggests faster web queries. Supervisor stores results.

Requirements

1. Implement a storage/retrieval system (e.g., dictionary with "query: solar, output: 9/10").
2. Simulate the six agents and Supervisor.
3. Add real-time web extraction (e.g., scrape a site or use an API like Google Search).
   Example: Fetch "solar panel efficiency" from a renewable energy blog.
4. Show feedback-driven improvement over 2–3 cycles.
5. Demo processing a sample input (e.g., "urban energy solutions").

Constraints

- Use any language/framework (e.g., Python with BeautifulSoup or requests for web scraping).
- Prioritize functionality over optimization.
- Keep it simple (e.g., text-based output).

Deliverables

1. Working prototype code.
2. Short demo showing input processing, web extraction, and improvement.
   Example Output: "Cycle 1: Solar panels (8/10, web: $200/unit); Cycle 2: Solar windows (9/10, web: urban trend)."
3. Brief explanation of your design.

Milestones

Milestone 1: Build Core Agent System and Memory

- **Goal**: Set up the Supervisor, specialized agents, and basic storage/retrieval.
- **Tasks**:
  - Implement a supervisor agent to assign tasks (e.g., simple queue or function calls).
  - Code basic versions of the six agents (Generation, Reflection, Ranking, Evolution, Proximity, Meta-review) with minimal logic (e.g., Generation outputs a hypothesis like "new treatment method").
  - Create a storage/retrieval mechanism (e.g., dictionary storing "query: disease cure, output: hypothesis A").
- **Deliverable**: Supervisor can call agents and store/recall a test input/output pair.

Milestone 2: Add Web Extraction and Agent Collaboration

- **Goal**: Integrate real-time web data and connect agents for basic collaboration.
- **Tasks**:
  - Add web scraping or API calls (e.g., fetch "latest treatment studies" from a site using requests or BeautifulSoup).
  - Enhance agents to use web data (e.g., Ranking scores with study results, Evolution refines with trends).
  - Link agents via Supervisor (e.g., Generation → Reflection → Ranking).
- **Deliverable**: System processes an input with web data and shows agent handoffs.

Milestone 3: Enable Iteration and Demo Prep

- **Goal**: Achieve iterative improvement and finalize the prototype.
- **Tasks**:
  - Implement a feedback loop (e.g., Evolution refines Ranking's top output, Proximity recalls past data).
  - Run 2–3 cycles to show improvement (e.g., "therapy A: 7/10" → "combined therapy: 8/10").
  - Prepare a demo (e.g., console output of cycles) and brief design notes.
- **Deliverable**: Demo shows input processed with web data, stored, and improved over cycles.

Evaluation Criteria

- **Creativity (20%)**: Innovative approach.
- **Functionality (30%)**: Core features, including web extraction, work.
- **Clarity (20%)**: Easy to follow.
- **Execution (20%)**: Code quality.
- **Impact (10%)**: Real-world potential.

Bonus Points

- Visualize agent interactions (e.g., log: "Web → solar data → Ranking: 8").
- Retain preferences across inputs (e.g., "solar preferred from last query").