Reflexion: Language Agents with Self-Reflection

2203
PDF Document · 591 KB

Other Approaches (previous studies , compared to Reflection) mentioned in the paper :

- **Self-Refine [15] & Pryzant et al. [21]:** These methods focus on **iterative self-improvement through self-evaluation**, where the LLM refines its output based on given constraints (e.g., "write this more positively").
    - **Limitation:** Primarily restricted to **single-generation** reasoning tasks, meaning they don't handle multi-step or sequential decision-making well.
    -
    - **Paul et al. [20]:** Utilize **fine-tuned "critic models"** to provide intermediate feedback during reasoning trajectories.
    - **Xie et al. [27]:** Employ **stochastic beam search over actions** for more efficient decision-making, leveraging self-evaluation for foresight.
    - **Yoran et al. [31] & Nair et al. [16]:** Use **"decider models"** to reason over multiple generated outputs.
    - **Kim et al. [10]:** Implement a **retry pattern** over a fixed number of steps but **without an explicit evaluation step** for learning from failures.
    - **Goodman [9]:** Involves a **qualitative evaluation step** to propose optimizations for previous generations

How it works:
Instead of changing the agent's "brain" (weights), Reflexion agents learn by:

1 Doing a task.
2 Getting feedback (if they did well or poorly).
3 Thinking about their mistakes in words (like writing in a diary).
4 Saving these "thoughts" in a memory.
5 Using these past "thoughts" to make **better decisions next time**.

Why it's good:

- It's flexible – works with different kinds of feedback.
- It makes agents much better at various tasks, like coding.
- For example, it made a coding agent better than GPT-4 (91% vs 80% success rate on a tough coding test!).

In short: Reflexion helps AI agents learn from their past experiences and mistakes by talking to themselves and remembering what they learned, making them smarter for future tasks

Reflexion: Learning from Mistakes through Verbal Feedback

Problem: Large Language Models (LLMs) used as agents struggle to learn quickly from trial-and-error, as traditional methods (like Reinforcement Learning) need huge amounts of data and expensive model changes.

Proposed Solution : "verbal reinforcement" – meaning, we give the agent feedback in words, not just numbers.

How it works :

- The agent gets a simple result (like "success" or "fail").
- Reflexion **changes this simple result into a text summary** (a "verbal feedback").
- This text summary is added to the LLM's context for the **next attempt/episode**.
- This "self-reflective feedback" acts like a **"semantic gradient"** – it tells the agent *how* to improve, not just *if* it failed.

Why this is powerful :
This helps agents learn from past mistakes to do better.

Advantages and Disadvantages Compared to Traditional RL

Advantages of Reflexion:

1 **Lightweight:** It doesn't require fine-tuning the Large Language Model (LLM) itself, making it efficient and easy to implement.
2 **Nuanced Feedback:** Unlike simple scalar rewards, it provides specific and interpretable linguistic feedback (e.g., targeted actions or corrections), allowing for a clearer understanding of successes and failures.
3 **Explicit Episodic Memory:** It stores past experiences and reflections in a human-readable text format, enabling the agent to explicitly understand and utilize its own history.
4 **Explicit Action Hints:** It offers clear guidance and hints for future actions, aiding the agent's decision-making process.

Disadvantages of Reflexion:

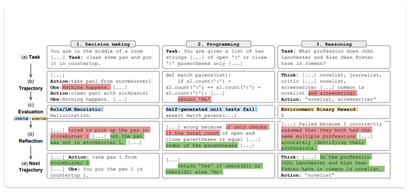
1 **Reliance on LLM Capabilities:** Its effectiveness heavily depends on the LLM's ability to self-evaluate and generate useful reflections.
2 **No Formal Guarantee of Success:** It does not provide a mathematical or formal guarantee of success.

Reflexion is a promising and interpretable new reinforcement learning paradigm that is expected to become even more powerful as LLM capabilities continue to improve.

Experiments

types of scenarios/tasks : Each item represents a **"scenario for experimentation"** or a **"test environment"** that requires the Reflexion agent to demonstrate specific capabilities.

- **(1) Decision-making tasks:** Scenarios where the agent needs to make a sequence of choices over a long path, similar to a game.
- **(2) Reasoning tasks:** Scenarios where the agent must use knowledge and logical deduction to answer complex questions.
- **(3) Programming tasks:** Scenarios where the agent writes and executes code, and interacts with external tools like compilers.



Experimental Methodology (Application of Reflexion):

Three methods for generating reflections were explored: simple binary feedback conversion, using pre-defined heuristics for common failures, and LLM self-evaluation (e.g., LLMs writing/executing unit tests in coding tasks).

- **Feedback Generation:** The agent receives feedback from the environment (e.g., success/failure, error messages) based on the outcome of its attempts (actions).
- **Verbal Reflection:** Based on this feedback, the LLM generates a **'reflection' text in natural language**. This reflection contains insights into "what went wrong, why it went wrong, and how to improve next time."
-
- **Memory Storage:** The generated verbal reflections are stored in the agent's **long-term memory**.
- **Iterative Learning:** In subsequent attempts, the agent utilizes these previously stored reflection memories as **additional context** to attempt better decision-making, reasoning, or code generation. This process is analogous to how humans learn from their failures.

Experimental Results and Comparison:

The performance of Reflexion agents was compared against strong baseline approaches in each task.
Consequently, Reflexion agents demonstrated significantly superior performance compared to existing baselines across all types of tasks.

- **AlfWorld:** Achieved an absolute 22% improvement in performance after 12 iterative learning steps.
- **HotPotQA:** Showed a 20% improvement in performance.
- **HumanEval:** Demonstrated an 11% improvement in performance (achieving SOTA with 91% pass@1 accuracy).

Figure 1: Reflexion works on decision-making 4.1, programming 4.3, and reasoning 4.2 tasks.

Related work on reasoning and decision-making

Approach	Self reflection	Hidden constraints	Decision making	Binary reward	Memory
Self-refine [15]	✓	✗	✗	✗	✗
Beam search [27]	✓	✗	✓	✗	✗
Reflexion (ours)	✓	✓	✓	✓	✓

Related work on programming

Approach Test execution	Test execution	Debugging	Self-generated tests	Multiple languages	Self-reflection
AlphaCode [14]	✓	✗	✗	✓	✗
CodeT [5]	✓	✗	✓	✗	✗
Self-debugging [7]	✓	✓	✗	✗	✗
CodeRL [13]	✓	✓	✗	✓	✗
Reflexion (ours)	✓	✓	✓	✓	✓

★ this paper quantitatively demonstrated how agents learn from failures and improve performance through verbal reflection, by selecting tasks of varying complexity levels and applying the Reflexion framework to each.

Reflexion provides a more holistic and dynamic approach where the LLM agent **continuously learns and evolves by internally reflecting on its past failures and accumulating that knowledge for future improvements.**

Reflexion's Unique Contribution:
Reflexion enhances many of these existing concepts by introducing **self-reflection as a continuous learning mechanism**.

- **Core Difference:** Unlike previous methods that might focus on single-shot refinement, limited retries, or external critic models, Reflexion builds a **"persisting memory of self-reflective experiences."**
- **Impact:** This memory allows the agent to:
- **Identify its own errors** over time.
- **Self-suggest lessons to learn from its mistakes.**
-

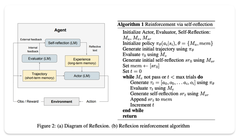


Figure 2: (a) Diagram of Reflexion. (b) Reflexion reinforcement algorithm

Algorithm 1 Reinforcement via self-reflection

Initialize Actor, Evaluator, Self-Reflection:
M_a, M_e, M_{sr}
Initialize policy $\pi_\theta(a_i|s_i), \theta = \{M_a, mem\}$
Receive initial observation y_0
Evaluate τ_0 using M_e
Generate initial self-reflection sr_1 using M_{sr}
Set $mem \leftarrow [sr_0]$
Set $t = 0$
while M_e not pass or $t <$ **max trials do**
  Generate $\tau_t = [a_0, o_0 \ldots, a_i, o_i]$ using π_θ
  Evaluate r_t using M_e
  Generate self-reflection sr_t using M_{sr}
  Append sr_t to mem
  Increment t
end while
return

Process

```
A[Start] --> B{Begin Trial t = 0};

B --> C[Actor: Generate trajectory τt by interacting with environment];

C --> D[Evaluator: Evaluate τt and compute scalar reward rt (rt = Me(τt))];

D --> E{Evaluator: Is τt correct?};

E -- Yes --> F[End: Optimal trajectory τt found];

E -- No --> G[Self-Reflection: Analyze {τt, rt} to produce verbal feedback srt];

G --> H[Memory (mem): Append srt to mem (bound by max Ω experiences)];

H --> I[Actor: Utilize srt from mem to inform generation of next trajectory τt+1];

I --> J{Increment Trial t = t+1};

J --> C;
```

Reflection

★

Memory
Core components of the Reflexion process are the notion of short-term and long-term memory.

- **Inference Time Decision-Making:** At inference time, the Actor conditions its decisions on both short-term and long-term memory. This is similar to how humans recall fine-grained recent details while also remembering distilled, important experiences from long-term memory.
- **Short-Term Memory (STM):** In the Reinforcement Learning (RL) setup, the **trajectory history** (the sequence of actions and observations from the current trial) serves as the short-term memory.
- **Long-Term Memory (LTM):** The outputs from the **Self-Reflection model** (the generated verbal reflections) are stored in long-term memory.
- **Combined Advantage:** These two memory components work together to provide context that is both **specific** (from the short-term trajectory) and also **influenced by lessons learned over several trials** (from the long-term reflections). This combined memory mechanism is a key advantage of Reflexion agents over other LLM action choice works.

Iterative Learning:
the agent can leverage its past experiences (stored in mem) to adapt its decision-making approach at time t by choosing action a'_i (the suggested alternative). This iterative process of trial, error, self-reflection, and persisting memory enables the agent to rapidly improve its decision-making ability in various environments by utilizing informative feedback signals.

The Self-Reflection model, instantiated as an LLM, plays a crucial role in the Reflexion framework by generating verbal self-reflections to provide valuable feedback for future trials.

- **Input:** Given a sparse reward signal (like a simple binary success/fail status), the current trajectory (the sequence of actions and observations), and its persistent memory mem (where past reflections are stored),
- **Output:** The self-reflection model generates **nuanced and specific feedback**. This feedback is **more informative than simple numerical (scalar) rewards** and is then stored in the agent's memory (mem).

Role of the Evaluator:

- **Check Quality:** The Evaluator checks how well the Actor's actions or text (called a 'trajectory') worked. It looks at the 'quality' of what was done.
- **Give Score:** Based on its check, it gives a 'reward score'. This score is a number that tells the agent how good its action was.

1. Role of the Actor:

- **Action Generation:** The Actor is an LLM. Its job is **to look at the agent's current 'state' and then create the necessary 'text' (like thoughts or plans) and 'actions' (like calling external tools or writing code).**
- **RL's 'Policy' Role:** It acts like the 'policy' ($\pi\theta$) in traditional Reinforcement Learning (RL). This means it decides what the agent should do next, given the current situation.
- **Sampling:** Following this policy, it picks a specific action (at) .After the action, it gets a new 'observation' (Otot) from the environment as a result. This cycle keeps repeating.

AlfWorld experiment for Reflexion

AlfWorld Environment:

- **Text-based, multi-step decision-making tasks:** (e.g., finding, moving, manipulating objects).
- **Self-evaluation is essential:** The environment only indicates task completion, requiring the agent to self-evaluate its progress.
- **Action Generator:** ReAct from Yao et al. [30] was used to generate actions.

, Self-Evaluation Methods:

- **Natural Language Classification using LLM:** The LLM directly evaluates trajectories and provides feedback.
- **Hand-written Heuristics:**
- Triggers self-reflection if the same action/response repeats more than 3 times (indicating inefficiency or "hallucination").
- Triggers self-reflection if the number of actions exceeds 30 (indicating inefficient planning).
-
- **Role of the Self-Reflection Module:** This module analyzes failed trajectories and reward information to generate specific verbal feedback (sr^t) about *what went wrong and why*, not just that it was wrong. This feedback acts as a crucial "self-hint" to prevent the agent from repeating the same mistakes in future attempts.

Comparison: Reflexion vs. Baseline (ReAct-only):

- **Baseline (ReAct-only):** If self-reflection is deemed necessary, the environment is reset, and a new trial begins *without* the self-reflection process.
- **Reflexion Applied:** If self-reflection is needed, the agent **analyzes its mistakes through self-reflection, updates its memory**, then resets the environment and starts a new trial.

Memory Management:

- **Strategic Memory Storage:** To stay within LLM context limits, the memory stores only the **last 3 self-reflections (experiences)**. This is a strategic choice to keep the most relevant and useful information.
- **Contribution to Long-term Learning:** This memory allows the agent to effectively use past lessons, maintain long-term context, and anticipate similar errors in complex problem-solving.
- **Initial Setup:** Domain-specific few-shot examples were provided to prevent syntactic errors in the initial agent's behavior.

Key Results:

- **Significant Performance Boost: ReAct + Reflexion** successfully completed **130 out of 134 AlfWorld tasks**, showing **markedly superior performance** compared to the ReAct-only approach.
- **Error Detection & Handling:** Reflexion effectively detected and managed "hallucinations" and inefficient planning using its heuristics.
- **Continuous Learning:** ReAct + Reflexion demonstrated the ability to learn and solve additional tasks over 12 consecutive trials.
- **Limitations of Baseline:** In contrast, the ReAct-only approach showed a performance plateau, with improvement stopping between trials 6 and 7. The "hallucination" issues and lack of long-term improvement in ReAct-only strongly highlight the need for self-correction mechanisms like Reflexion.

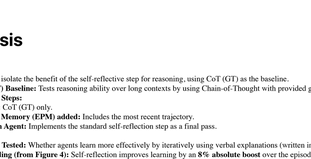


Figure 3: (a) AlfWorld performance across 134 tasks showing cumulative proportions of solved tasks using self-evaluation techniques of (Heuristic) and (GPT) for binary classification. (b) Classification of AlfWorld trajectories by reason of failure.

Reasoning: HotPotQA

- **Reasoning-Only Setup:**
    - Used **Reflexion + Chain-of-Thought (CoT)**.
    - Agent was given the ground truth context (Cgt) to isolate
    - 6-shot prompting for CoT.

- **Holistic Q&A Setup:**
    - Used **Reflexion + ReAct**.
    - Agent could retrieve context using a Wikipedia API and infer answers with step-by-step thinking.
    - 2-shot prompting for ReAct.
    -

- **Self-Reflection Implementation:**
    - 2-shot prompting for self-reflection.
    - Used **exact match answer grading** for a binary success signal between trials.
    - Self-reflection loop amplified this binary signal, similar to AlfWorld, with a memory size of **3 experiences**.
    - Reflexion agents retried failed tasks until 3 consecutive failures on that task.

- **Key Results:**
    - **Reflexion significantly outperformed all baseline approaches** (ReAct-only, CoT-only, CoT (GT)-only) across multiple learning steps.
    - Baseline approaches **failed to probabilistically improve** on any tasks: once failed, they couldn't solve them later.
    - Even **CoT (GT)** (with ground truth context) failed on **39% of questions**.
    - **Reflexion improved accuracy by 14%** for CoT (GT) by helping the agent correct mistakes *without* needing access to the ground truth answer during the correction process.
    - This shows Reflexion's ability to **enhance reasoning and self-correction** even in complex Q&A tasks.

Figure 4: Chain-of-Thought (CoT) and ReAct. Reflexion improves search, information retrieval, and reasoning capabilities on 100 HotPotQA questions. (a) Reflexion ReAct vs Reflexion CoT (b) Reflexion CoT (GT) for reasoning only (c) Reflexion vs episodic memory ablation.

Analysis

- **Goal:** To isolate the benefit of the self-reflective step for reasoning, using CoT (GT) as the baseline.
- **CoT (GT) Baseline:** Tests reasoning ability over long contexts by using Chain-of-Thought with provided ground truth context.
- **Ablation Steps:**
1 **Baseline:** CoT (GT) only.
2 **Episodic Memory (EPM) added:** Includes the most recent trajectory.
3 **Reflexion Agent:** Implements the standard self-reflection step as a final pass.
-
- **Intuition Tested:** Whether agents learn more effectively by iteratively using verbal explanations (written in the first person).
- **Key Finding (from Figure 4):** Self-reflection improves learning by an **8% absolute boost** over the episodic memory learning advantage.
- **Conclusion:** This result supports the argument that "refinement-only" approaches are **less effective** than "self-reflection-guided refinement" approaches.

Benchmark + Language	Prev SOTA Pass@1	SOTA Pass@1	Reflexion Pass@1
HumanEval (PY)	65.8 (CodeT [5] + GPT-3.5)	80.1 (GPT-4)	**91.0**
HumanEval (RS)	–	60.0 (GPT-4)	**68.0**
MBPP (PY)	67.7 (CodeT [5] + Codex [6])	**80.1** (GPT-4)	77.1
MBPP (RS)	–	70.9 (GPT-4)	**75.4**
Leetcode Hard (PY)	–	7.5 (GPT-4)	**15.0**

Table 1: Pass@1 accuracy for various model-strategy-language combinations. The base strategy is a single code generation sample. All instruction-based models follow zero-shot code generation.

Benchmark + Language	Base	Reflexion	TP	FN	FP	TN
HumanEval (PY)	0.80	**0.91**	0.99	0.40	0.01	0.60
MBPP (PY)	**0.80**	0.77	0.84	0.59	0.16	0.41
HumanEval (RS)	0.60	**0.68**	0.87	0.37	0.13	0.63
MBPP (RS)	0.71	**0.75**	0.84	0.51	0.16	0.49

Table 2: Overall accuracy and test generation performance for HumanEval and MBPP. For Rust, HumanEval is the hardest 50 problems from HumanEval Python translated to Rust with MultiPL-E [4]. TP: unit tests pass, solution pass; FN: unit tests fail, solution pass; FP: unit tests pass, solution fail; TN: unit tests fail, solution fail.

Conclusion

Reflexion, an approach that leverages verbal reinforcement to teach agents to learn from past mistakes. We empirically show that Reflexion agents significantly outperform currently widely-used decision-making approaches by utilizing self-reflection. In future work, Reflexion could be used to employ more advanced techniques that have been thoroughly studied in traditional RL settings, such as value learning in natural language or off-policy exploration techniques.

Sample "Reflection"code

Assumptions:

- **Ma:** Actor (An LLM that generates actions).
- **Me:** Evaluator (An LLM or external system that assesses the outcome or trajectory of the generated actions).
- **Msr:** Self-Reflection Model (An LLM that generates reflection texts).
- **mem:** The agent's self-reflection memory (in the form of a list).
- **max_trials:** The maximum number of attempts/trials.
- **Me.post():** Indicates whether the Evaluator signifies a 'success' state (e.g., passing a coding problem).

```
import random # Used for simulated random outcomes in the example

# --- Define Mock/Abstract Components ---

class Actor:
    """Represents the Actor LLM (Ma) that generates actions."""
    def __init__(self, llm_parameters=None):
        self.llm_parameters = llm_parameters # Simulates LLM configuration (e.g., model name, temperature)

    def generate_action(self, state: str, memory: list) -> str:
        """
        Generates the next action based on the current state and the agent's memory.
        In a real implementation, this would involve prompting an LLM.
        """
        print(f"Actor: Current state is '{state}'. Reflecting on memory: {memory}")
        # Example: Generate a hypothetical action string
        # The policy πθ(ai |si) is parameterized by Ma and the memory (mem)
        action = f"Attempt based on current state '{state}' and " \
                 f"past reflection '{memory[-1] if memory else 'no previous insights'}'"
        return action

class Evaluator:
    """Represents the Evaluator (Me) that assesses the outcome of a trajectory."""
    def __init__(self):
        self._passed = False # Initial state: task not passed
        self.trial_count = 0

    def evaluate_trajectory(self, trajectory: str) -> str:
        """
        Evaluates the given trajectory and returns a feedback string.
        In a real scenario, this would involve interacting with an external environment
        (e.g., running code, checking game state) and analyzing its output.
        """
        self.trial_count += 1
        print(f"Evaluator: Evaluating trajectory: '{trajectory}'")

        # Simulated evaluation logic:
        # 50% chance to pass randomly, or higher chance after a few trials,
        # or if trajectory explicitly contains "correct" (for demonstration)
        if "correct" in trajectory.lower() or (self.trial_count > 2 and random.random() < 0.7):
            self._passed = True
            feedback = "SUCCESS: The task was completed correctly."
        else:
            feedback = f"FAILURE: The task failed. Reason: {random.choice(['syntax error', 'logic error', 'timeout', 'incorrect output'])}. Trajectory was: '{trajectory}'"
        return feedback

    def is_passed(self) -> bool:
        """Returns True if the task has been successfully passed, False otherwise."""
        return self._passed

class SelfReflectionModel:
    """Represents the Self-Reflection Model (Msr) that generates reflections."""
    def __init__(self):
        pass

    def generate_reflection(self, feedback: str, trajectory: str, memory: list) -> str:
        """
        Generates a self-reflection text based on the feedback, the executed trajectory,
        and potentially the past memory.
        In a real implementation, this would involve prompting an LLM with the feedback.
        """
        print(f"SelfReflectionModel: Generating reflection for feedback: '{feedback}'")
        if "SUCCESS" in feedback:
            reflection = f"Successfully completed the task. Key learning from this success: {feedback}. Previous trajectory: '{trajectory}'"
        else:
            # For failures, suggest actionable insights
            reflection = f"Failed. Need to re-evaluate approach. Feedback: {feedback}. Previous trajectory: '{trajectory}'. " \
                         f"Consider trying a different strategy or debugging {random.choice(['variable names', 'loop conditions', 'API calls', 'overall plan'])}."
        return reflection

# --- Reflexion Algorithm Implementation ---

def run_reflexion_agent(max_trials: int = 5):
    """
    Executes the learning loop for a Reflexion agent as described in the pseudocode.
    """
    # Initialize Actor, Evaluator, Self-Reflection: Ma, Me, Msr
    ma = Actor(llm_parameters={"model": "GPT-4", "temperature": 0.7}) # Ma
    me = Evaluator()                                                  # Me
    msr = SelfReflectionModel()                                       # Msr

    # Initialize policy πθ(ai |si), θ = {Ma, mem}
    # The policy is implicitly defined by how the Actor (Ma) uses the current state and memory.
    mem = []  # Agent's episodic memory buffer, initialized as empty

    # Generate initial trajectory using πθ
    print("\n--- Initial Trial (t=0) ---")
    current_state = "Starting the task from scratch."
    # The first action is generated without any prior reflection in memory
    initial_action = ma.generate_action(current_state, mem)
    # A simple representation of the trajectory (sequence of actions/observations)
    initial_trajectory = f"Initial attempt: '{initial_action}'"

    # Evaluate τ0 using Me
    initial_feedback = me.evaluate_trajectory(initial_trajectory)

    # Generate initial self-reflection sr0 using Msr
    initial_reflection = msr.generate_reflection(initial_feedback, initial_trajectory, mem)

    # Set mem ← [sr0]
    mem.append(initial_reflection)
    print(f"Memory after initial reflection: {mem}\n")

    # Set t = 0 (loop counter)
    t = 0

    # while Me not pass or t < max trials do
    # Continue trials as long as the task is not passed AND we haven't exceeded max_trials
    while not me.is_passed() and t < max_trials:
        t += 1 # Increment trial counter for the current iteration
        print(f"\n--- Trial {t} ---")

        # Generate τt = [a0, o0, . . . ai , oi ] using πθ
        # The Actor now uses the accumulated 'mem' (reflections) as part of its policy
        current_state = f"Continuing the task based on previous attempts. Trial count: {t}"
        action = ma.generate_action(current_state, mem)

        # In a real system, 'action' would be executed in the environment,
        # and 'observation' would be received, forming the full trajectory.
        # Here, we simplify the trajectory representation.
        trajectory = f"Action in trial {t}: '{action}'"

        # Check if task was already passed in a previous iteration (edge case)
        if me.is_passed():
            print("Task already passed, no new action needed for this iteration.")
            break # Exit the loop if already passed

        # Evaluate τt using Me
        feedback = me.evaluate_trajectory(trajectory)

        # Generate self-reflection srt using Msr
        reflection = msr.generate_reflection(feedback, trajectory, mem)

        # Append srt to mem
        mem.append(reflection)
        print(f"Memory after trial {t} reflection: {mem}\n")

    # end while

    # return (Final results summary)
    print("\n--- Final Results ---")
    if me.is_passed():
        print(f"Task completed successfully after {t} trials!")
    else:
        print(f"Task not completed within {max_trials} trials.")
    print(f"Final Memory: {mem}")
    # Display the last feedback received, or initial if only one trial occurred
    print(f"Final Feedback: {initial_feedback if t==0 else feedback}")

# Execute the Reflexion agent simulation
if __name__ == "__main__":
    run_reflexion_agent(max_trials=5)
```