Agentic Research Assistant with RL Feedback using CrewAI

Sinon
Building Agentic Systems Assignment

November 24, 2025

Abstract

This project implements an agentic AI research assistant using the CrewAI platform and its visual Studio. The system follows a multi-agent architecture with a central Controller agent that delegates subtasks to specialized agents for web research, evidence analysis, and technical writing. It integrates three built-in tools (web search, web scraper, file reader) and one custom tool for citation formatting.

Simple reinforcement-learning-style feedback is applied by computing a reward signal based on the number of sources found and the final answer length; this reward is logged and can be used to iteratively refine prompts and configuration. The system is evaluated on five test cases ranging from straightforward factual queries to edge-case nonsense input.

Overall, the assistant successfully generates structured Markdown research reports for meaningful queries, demonstrates robust orchestration and tool usage, and clearly exposes limitations around hallucinations for nonsensical inputs—highlighting the need for explicit guardrails in agentic systems.

Contents

1 System Overview

1.1 Application Domain

Domain: Research Assistant.

User goal: The user asks a complex research question (for example, "How will electric vehicle adoption impact global CO_2 emissions over the next 20 years?"), and receives a structured, referenced research report in Markdown.

Outputs:

- Title
- Abstract
- Table of Contents
- Main sections with key arguments
- Key takeaways
- Limitations and future work
- References section (from formatted citations)
- Suggested filename

1.2 High-Level Workflow

1. The user enters a research query in CrewAI Studio.
2. A *Controller Planning Task* (ControllerAgent) decomposes the query into 3–6 sub-questions and a short action plan.
3. The *Research Task* (ResearchAgent) uses web tools to search and scrape relevant pages and returns a structured list of sources.
4. The *Analysis Task* (AnalysisAgent) summarizes and synthesizes key findings from the sources into a Markdown analysis section.
5. The *Writer Task* (WriterAgent) uses the analysis, sources JSON, and plan context plus the custom citation tool to produce a final Markdown research report.
6. The Controller conceptually computes a simple reward based on system output (for example, number of sources, length of output) and logs it as part of memory for future improvements.

2 Architecture

2.1 Architecture Diagram

Figure 1 shows the overall architecture and control flow between the agents and tools.

3 Agent Design

The CrewAI project defines four main agents: a controller/orchestrator and three specialists.

3.1 ControllerAgent – "Research Orchestrator"

- **CrewAI Studio Agent Role:** Research Orchestrator
- **Model:** `gpt-4o-mini`
- **Tools:** None directly (relies on other agents' tools).
- **Primary Task:** Controller Planning Task

Responsibilities

- Accepts `user_query` as input.
- Decomposes the query into 3–6 sub-questions.
- Produces a short plan that:
  - Lists sub-questions.
  - Assigns which parts belong to ResearchAgent, AnalysisAgent, and WriterAgent.
  - Mentions any obvious risks (for example, limited data, controversial topic).
- Passes the plan and sub-questions forward to subsequent tasks.

Success Criteria

- The plan is clear, concise, and contains actionable steps.
- Sub-questions are non-overlapping and cover the main aspects of the query.

Error Handling

If the query is empty or extremely vague, the plan includes a note such as:
"The user query is unclear; recommend asking for clarification or narrowing scope."

Memory Usage

The agent logs:

- The original user query.
- Decomposed sub-questions.
- The high-level plan.

This context is reused by the Analysis and Writer agents to maintain coherence.

3.2 ResearchAgent – "Web Research Specialist"

- **CrewAI Studio Agent Role:** Web Research Specialist
- **Model:** `gpt-4o-mini`
- **Tools (built-in):**
  - **SerplyWebSearchTool** – search engine wrapper (requires `SERPLY_API_KEY`).
  - **ScrapeWebsiteTool** – fetches and extracts text from web pages.
- **Task:** Research Task

Inputs

- Full `user_query`.
- Sub-questions and plan from the Controller.

Behavior

- Performs web search for the main query and each sub-question.
- Uses the scraper tool to extract important content from top results.
- Filters out spammy or non-relevant pages as much as possible using prompt instructions.

Output Format

```
{
  "sources": [
    {
      "title": "...",
      "url": "https://example.com",
      "snippet": "Short description or key excerpt..."
    },
    ...
  ]
}
```

Success Criteria

- Typically returns 3–5 relevant sources (more are allowed for complex topics).
- Snippets are actually about the topic, not random website boilerplate.

Fallback Behavior

If no relevant pages are found, the agent returns an empty or very small `sources` list and a note that data is limited.

Memory Usage

The `sources` JSON is stored in the run history and reused by the AnalysisAgent and Writer-Agent.

3.3 AnalysisAgent – "Evidence Synthesizer and Analyst"

- **CrewAI Studio Agent Role:** Evidence Synthesizer and Analyst
- **Model:** `gpt-4o-mini`
- **Tools (built-in):**
  - **FileReadTool** – to read structured source content if saved or available.
- **Task:** `analysis_task`

Inputs

- `sources` JSON from the ResearchAgent.
- Plan and sub-questions from the Controller.

Behavior

- Reads the sources list and extracts key findings for each sub-question.
- Produces a Markdown block with:
  - ## **Key Findings** — bullet points summarizing main ideas.
  - ## **Synthesis** — 10–15 sentence narrative combining the evidence.
- Adds notes about agreements or conflicts between sources.

Handling Empty Sources

If `sources` is empty:

- The agent explicitly states that no reliable sources were found.
- It suggests possible reasons (niche topic, nonsense term, typo).

Success Criteria

- Clear, non-repetitive summaries.
- Explicit attribution to sources (for example, "One study from IEA suggests…").

Memory Usage

The Markdown analysis and any "no sources" flags are stored as part of the run history.

3.4 WriterAgent – "Technical Research Writer"

- **CrewAI Studio Agent Role:** Technical Research Writer
- **Model:** `gpt-4o-mini`
- **Tools:**
  - **FileReadTool** – to read analysis output when saved as a file.
  - **CitationBuilderTool** – custom Python tool (`src/agentic_research_assistant_with_rl_feedback/t`
- **Task:** `writer_task`

Inputs

- Markdown analysis from the AnalysisAgent.
- `sources` JSON from the ResearchAgent.
- Plan and sub-questions from the Controller.

Behavior

- Uses the custom CitationBuilderTool to transform sources into a clean References section.
- Produces the final Markdown research report with sections:
  - **# Title**
  - **## Abstract** (3–5 sentences)
  - **## Table of Contents**
  - **## Main Sections** (based on key findings and sub-questions)
  - **## Key Takeaways** (3–7 bullets)
  - **## Limitations and Future Work**
  - **## References** (from CitationBuilderTool)
- Suggests a filename such as `research_report_<short_topic>.md`.

Success Criteria

- All required sections appear.
- References are well formatted and aligned with the underlying sources.
- Output is coherent and readable to a non-expert.

Fallback Behavior

If `sources` is empty:

- Adds a statement in References: "No credible sources were found for this query."
- Abstract and main sections emphasize uncertainty and the lack of data.

4 Tools and Integration

4.1 Built-in Tools

At least three built-in tools are integrated.

4.1.1 SerplyWebSearchTool

- Used by the Web Research Specialist in the Research Task.
- **Purpose:** Perform web search from within CrewAI.
- **Configuration:** Requires environment variable `SERPLY_API_KEY`.
- **Role in System:** Primary web search engine; returns URLs and brief snippets.

4.1.2 ScrapeWebsiteTool

- Used by the Web Research Specialist.
- **Purpose:** Extract main content from selected URLs.
- **Role in System:** Allows the agent to gather detailed text for better summaries and fact extraction.

4.1.3 FileReadTool

- Used by the Evidence Synthesizer and Analyst and the Technical Research Writer.
- **Purpose:** Read intermediate content saved to files during the workflow.
- **Role in System:** Provides data processing and transformation capabilities and enables agents to share structured content across steps.

4.2 Custom Tool – CitationBuilderTool

File: `src/agentic_research_assistant_with_rl_feedback/tools/custom_tool.py`

Purpose

Convert the `sources` JSON (list of {`title`, `url`, `snippet`}) into a consistent references block for the final report.

Inputs

Python list of dictionaries or a JSON string with structure:

```
[
  {"title": "Example Study", "url": "https://example.com", "snippet": "..."},
  ...
]
```

Output

A formatted string, for example:

```
## References
1. Example Study. Link: https://example.com
2. Another Article. Link: https://another.com
```

Input Validation and Error Handling

- If the input is `None`, an empty list, or not valid JSON, the tool returns:

```
 References
No valid sources were available to build references.
```

- If a source lacks `title` or `url`, the tool skips it or replaces missing values with "Unknown Title" / "No URL" to avoid crashes.

How it Enhances Performance

- Offloads citation formatting from the LLM, reducing hallucinated titles or links.
- Enforces a consistent structure for references across runs.
- Makes it easier to parse references later for evaluation or downstream use.

5 Orchestration and Memory

5.1 Orchestration Flow in CrewAI Studio

The system is implemented in the CrewAI Studio Visual Editor as a chain of tasks:

1. A **Trigger** starts the run when the user provides `user_query`.
2. **Controller Planning Task** (ControllerAgent) outputs a plan and sub-questions.
3. **Research Task** (ResearchAgent) uses search and scraper tools and returns `sources`.
4. **Analysis Task** (AnalysisAgent) creates a Markdown analysis based on `sources` and the plan.
5. **Writer Task** (WriterAgent) uses analysis, `sources`, and the CitationBuilderTool to generate the final Markdown report.

Flow type: sequential and hierarchical (sequential in terms of tasks; hierarchical in responsibility, with the Controller overseeing the specialists).

5.2 Memory Management

Short-Term Memory

Within a single run, outputs from each task (plan, `sources`, analysis) are passed as context to later agents.

Persistent Memory File

In the exported Python project, `knowledge/user_preference.txt` can be used to store information about user preferences (for example, preferred length or style).

RL Logging

After each run, the system conceptually tracks:

- Number of sources found.
- Final report length (characters or words).
- Derived reward (see Section 6).

6 Reinforcement Learning Concept

The project uses a simple reward-shaping scheme rather than a full RL algorithm.

Reward for Sources

$$r_{sources} = \{ 1 \, , iflen(sources) \geq 30, otherwise$$

Penalty for Overly Long Outputs

$$r_{length} = \{ -1, iflen(final\_report) > 1500characters0, otherwise$$

Final Reward

$$r_{final} = r_{sources} + r_{length}.$$

Usage

- After each research session, the system logs r_{final}.
- Over multiple runs, it is possible to compute a running average reward.
- This reward guides manual adjustments:
  - If rewards are often 0 or -1, tighten prompts to limit verbosity or encourage the ResearchAgent to search more effectively.
  - If rewards are consistently high, the current prompts and configurations are working well.

The implementation does not include full policy-gradient RL; instead, RL concepts are used to inform human-in-the-loop prompt tuning, which is appropriate for the assignment's scope.

7 Challenges and Solutions

Challenge 1: Agent Coordination in CrewAI Studio

Issue: Early runs stopped mid-pipeline or kept "running" due to misconfigured tasks or loops.
Solution:

- Simplified the pipeline to a strictly sequential flow.
- Carefully configured each task's description and expected output so the LLM knows exactly what to produce and pass on.

Challenge 2: Running the Exported Project in VS Code

Issue: Encountered `ModuleNotFoundError` for the project package and missing environment variables.
Solution:

- Activated the correct virtual environment.
- Set `PYTHONPATH` to `src`.
- Created a `.env` file with `OPENAI_API_KEY` and `SERPLY_API_KEY`.

Challenge 3: Handling Nonsense Queries

Issue: For an input like "asdkjhaskdjh quantum potato protocol 9999?", the system still generated a confident research report (hallucination).
Solution (partial):

- Documented this behavior as a known limitation in the evaluation.
- Proposed future work to add guardrails so ResearchAgent and AnalysisAgent explicitly detect when no valid sources exist and instruct WriterAgent not to fabricate content.

8 System Performance and Limitations

8.1 Performance Summary

For well-formed, real-world questions, the system:

- Returns coherent, structured Markdown reports.
- Uses web search and sources effectively.
- Maintains topic focus and context across agents.

For edge cases:

- Empty or very vague queries produce weaker results or need clarification.
- Nonsense queries can still trigger hallucinated reports (Test Case T5).

8.2 Key Limitations

- **Hallucinations:** There is no hard check that sources actually mention the queried concept, which leads to misbehavior for nonsense inputs.
- **Source Quality:** The system does not deeply rank or verify sources (for example, peer-reviewed articles versus blog posts).
- **RL Feedback Not Fully Automated:** The reward is conceptual and not yet used to automatically update prompts or parameters.
- **Tool Dependency:** The pipeline requires working API keys (`OPENAI_API_KEY`, `SERPLY_API_KEY`). Breaking changes in the tools or APIs could affect behavior.

8.3 Proposed Future Improvements

- Add explicit "no valid sources found" signals and logic across agents.
- Introduce simple confidence scores (for example, based on number and agreement of sources).
- Log rewards to a file (for example, `metrics.json`) and implement a small script to:
  - Adjust prompts when average reward drops.
  - Explore prompt variants or different tool configurations automatically.
- Extend custom tools (for example, keyword highlighter, outline generator, or source-quality classifier).

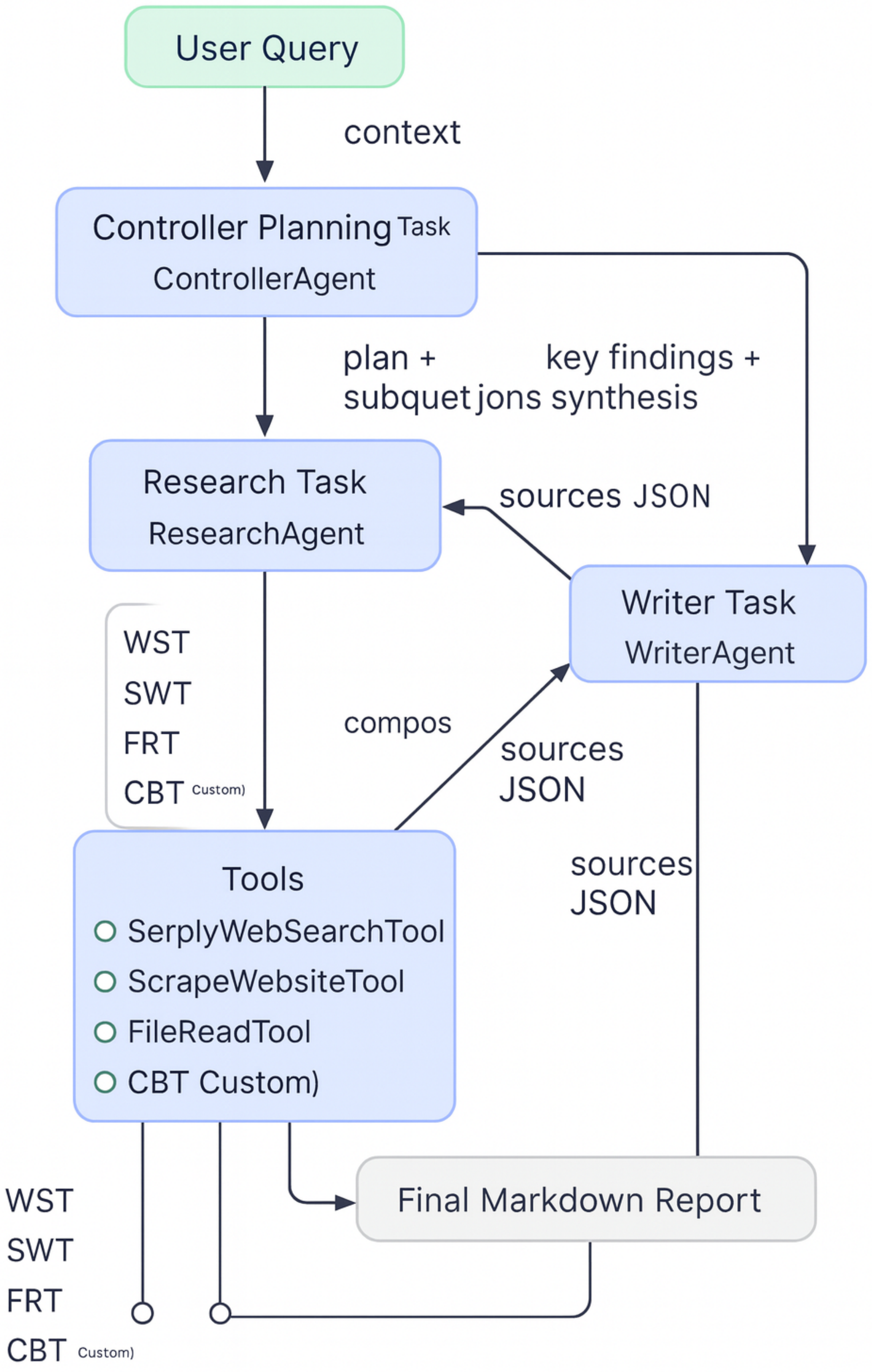

Figure 1: High-level architecture of the Agentic Research Assistant with RL Feedback.