Seeker-o1: A Hybrid AI Agent System with Dynamic Execution Modes, Advanced Memory and LLM-Powered Planning

Ibrahim Rayamah

01/05/2025

Abstract

Large Language Models (LLMs) have enabled the development of AI agents capable of complex reasoning and task execution. However, building flexible and efficient agent systems remains challenging. This paper introduces Seeker-o1, a command line based AI agent system designed for modularity and extensibility. Seeker-o1 features a hybrid agent architecture that dynamically switches between a direct LLM response mode for simple tasks and a multi-agent collaboration mode for complex problems, optimizing resource utilization. The framework incorporates an LLM-powered task planner for generating and adapting execution strategies, a dual-component memory system (short-term volatile and long-term persistent), and a dynamic tool discovery and execution mechanism. We detail the architecture, implementation of key components like the Agent Orchestrator, Hybrid Agent, Task Planner, and Memory systems, and discuss its capabilities in handling diverse tasks including text processing, code execution, and image understanding. This project provides a great foundation for building and experimenting with autonomous agents.

1 Introduction

The pursuit of Artificial General Intelligence (AGI) has spurred significant research into autonomous agents capable of perception, reasoning, planning, and acting within complex environments [1, 2]. Recent advances in Large Language Models (LLMs) have dramatically accelerated progress, enabling agents that leverage the vast knowledge and reasoning capabilities embedded within these models [3]. Frameworks like LangChain [5], AutoGen [6], and Transformers Agents [4] have emerged to facilitate the development of these LLM-powered agents.

However, challenges remain in terms of handling of tasks. Simple tasks might be handled directly by an LLM, while complex, multi-step problems often benefit from decomposition and specialized tool use or collaborative reasoning patterns like ReAct [7]. Existing frameworks often adopt a fixed approach, potentially leading to inefficiency for simpler queries or insufficient capability for complex ones.

Seeker-o1 distinguishes itself through:

- **Hybrid Execution Model:** A core `HybridAgent` dynamically assesses task complexity and switches between direct LLM interaction and a multi-agent decomposition strategy involving specialized sub-agents (e.g., researcher, planner, executor, critic).
- **LLM-Powered Planning:** Integration of a `TaskPlanner` that utilizes LLMs to generate, execute, and adapt multi-step task plans.
- **Dual Memory System:** Combines volatile short-term memory (STM) for immediate context with persistent long-term memory (LTM) for retaining information across sessions.
- **Extensible Tool Ecosystem:** A modular system for integrating and discovering tools, allowing agents to interact with external resources and perform diverse actions (e.g., calculations, code execution, web search).

This paper details the architecture, implementation, and features of Seeker-o1. Section 2 discusses related work. Section 3 outlines the core architecture. Section 4 delves into the implementation details of key components. Section 5 highlights significant features. Section 6 concludes with potential applications and future directions.

2 Related Work

The field of AI agents, particularly those powered by LLMs, is rapidly evolving. Several frameworks provide foundational capabilities for agent development.

- **LangChain & LlamaIndex:** These libraries offer extensive components for building LLM applications, including agent abstractions, memory modules, and tool integrations. They provide flexibility but often require significant developer effort to orchestrate complex agent behaviors.
- **AutoGen:** Focuses on multi-agent conversations and collaboration patterns, enabling complex workflows through interactions between specialized agents [6]. Seeker-o1 incorporates multi-agent concepts within its `HybridAgent` but offers a dynamic switching mechanism not explicitly central to AutoGen's core design.
- **Transformers Agents:** Provides agents capable of interpreting natural language instructions and selecting appropriate tools from a predefined set [4]. Seeker-o1 builds on this concept with dynamic tool discovery and a more complex planning and execution engine.
- **AgentLite / Agents:** Explores lightweight agent frameworks and formalizes concepts like long-short term memory and multi-agent collaboration structures [4].

Seeker-o1 differentiates itself by combining the dynamic hybrid execution model (single vs. multi-agent based on complexity) with integrated LLM-based task planning and a clear separation of core components like the orchestrator, agents, memory, and tools within an extensible open-source structure as mentioned in previous paragraphs.

3 Architecture

Seeker-o1 employs a modular architecture designed for flexibility and extensibility. Below is a simplified representation of its core flow, focusing on how tasks are processed from input to output.

The diagram at **"figure: 1"** illustrates the primary workflow of Seeker-o1: - User Task: Initiates the process by providing input to the system. - Agent Orchestrator: Receives the task and coordinates the overall execution. - Hybrid Agent: Evaluates the task's complexity. For simple tasks, it directly produces a result (indicated by the curved arrow to "Final Result"). For complex tasks, it delegates to the Task Planner. - Task Planner: Decomposes complex tasks into actionable steps using an LLM. - Sub-Agents: Execute the plan, with roles like Researcher (gathers information) and Critic (evaluates outcomes) - Final Result: The output returned to the user.

While the diagram focuses on the core flow, the complete Seeker-o1 architecture comprises of the following: **Orchestrator** (`AgentOrchestrator`): Manages task routing and resource allocation. **Agents** (`BaseAgent`, `HybridAgent`, `ToolAgent`, `ReactAgent`): Handle task execution, with the `HybridAgent` switching between direct and multi-agent modes. **Task Planner** (`TaskPlanner`): Generates and adapts execution plans using LLMs. **Memory System** (`BaseMemory`, `ShortTermMemory`, `LongTermMemory`): Manages short-term and long-term context. **Tool Ecosystem** (`BaseTool`, `ToolCollection`): Enables interaction with external resources via modular tools. **Model Integration** (`ModelRouter`, `VisionModel`): Connects to LLMs and vision models for multimodal capabilities.

This modular design ensures the agent can adapt to a wide range of tasks while remaining extensible for enhancements.

4 Implementation Details

4.1 Hybrid Agent and Dynamic Execution

The `HybridAgent`'s core logic resides in its execute method. Task complexity is assessed heuristically by the `_assess_complexity` method. This method calculates a score (capped at 10.0) based on: identifying keywords associated with specific operations (e.g., `calculate`, `search`, `compare`, `run code`, `and`, `if`, each adding 1.0-2.5 points); task length (0.1 points per word); the count of special characters (0.2 points each); and the number of distinct implied tools based on keywords (`calculator`, `search`, `text`, `code`, adding 1.5 points per tool type). If complexity exceeds a configurable threshold (default not specified in reviewed code), the `_execute_multi_agent` method is invoked. This method orchestrates a sequence:

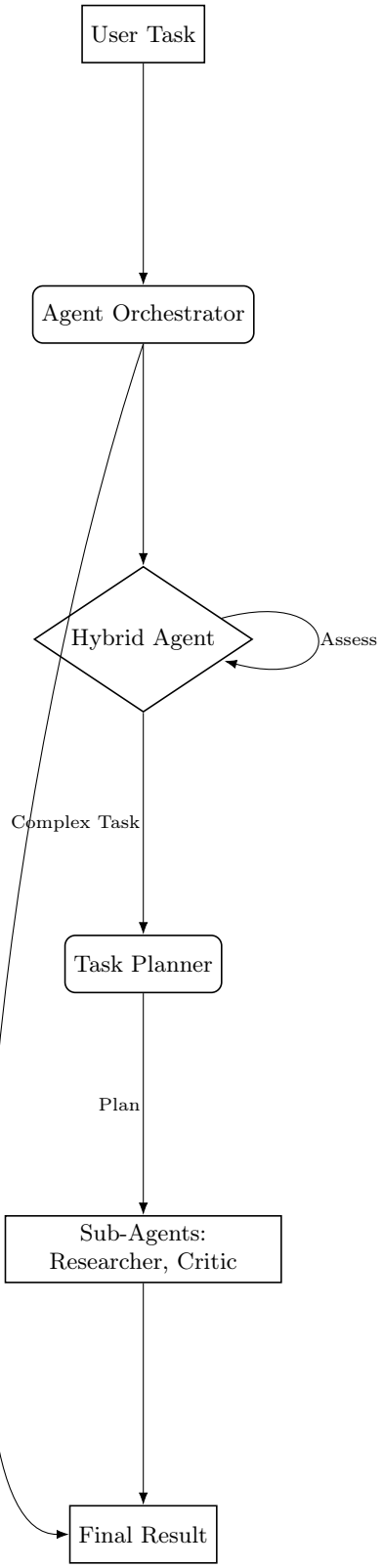

Figure 1: Seeker-o1 Framework Flow Diagram

the 'researcher' agent gathers context, the 'planner' outlines steps, the 'executor' performs actions (likely using tools via the inherited `ToolAgent` capabilities), and the 'critic' evaluates the result. For simpler tasks, it bypasses this multi-agent flow and directly calls the configured LLM through the `ModelRouter`. It also preprocesses tasks containing image paths using the `VisionModel`.

4.2 LLM-Powered Task Planning

The `TaskPlanner` leverages the `ModelRouter` to interact with a large language model. By default, the `ModelRouter` is configured to use the `openai` provider with the `gpt-4o` model and a temperature of `0.0`, although only the `OpenAIModel` class is registered by default. Its `create_plan` method constructs a detailed prompt instructing the LLM to break down the input task into a sequence of steps, specifying the tool and parameters for each step, based on a provided JSON schema. The `replan` method allows the agent to adapt the plan based on intermediate results or errors, providing feedback to the LLM to generate revised steps.

4.3 Dual Memory System

`ShortTermMemory` uses Python dictionaries and a `deque`, enforcing capacity via FIFO/LRU-like eviction and TTL constraints (`_prune_expired`). `LongTermMemory` stores each item as a JSON file (`_save_item`), managing persistence. Searches (`search`) in LTM operate in two modes: if `index_in_memory` is enabled, it searches an in-memory dictionary loaded from all storage files at initialization (`_load_index`); otherwise, it scans the storage directory, loading each JSON file individually. The matching logic (`_matches_query`) performs exact key-value comparisons between the item and the query, supporting nested key lookups via dot notation (e.g., `query={'metadata.type': 'summary'}`). Results are sorted by creation time, newest first.

4.4 Tool Discovery and Management

The `ToolCollection` uses `pkgutil.iter_modules` within its discover_tools method to dynamically find modules within the `seeker_o1.tools` package (or other specified packages). It inspects these modules for classes inheriting from `BaseTool` and registers them in `tool_classes`. When a tool is requested via get_tool, it instantiates the class if needed. execute_tool retrieves the tool instance and invokes its execute method.

5 Key Features

- **Adaptive Execution:** The hybrid agent model automatically adjusts its strategy based on task complexity, enhancing efficiency.
- **Intelligent Planning:** LLM-based planning allows for sophisticated task decomposition and adaptation beyond simple rule-based approaches.
- **Context Management:** The dual memory system balances rapid access to recent information with long-term persistence.
- **Extensibility:** Modular design and dynamic tool discovery make it easy to add new capabilities (tools, models, agents).
- **Multimodality:** Basic image understanding is supported via the `VisionModel`.
- **Open Source:** Facilitates community contributions, transparency, and customization.

6 Conclusion and Future Work

The Seeker-o1 project provides a flexible and powerful open-source system for developing AI agents. Its hybrid execution model, LLM-powered planning, and modular design offer a robust foundation for tackling a wide range of tasks. The dynamic complexity assessment allows for efficient resource allocation, while the extensible tool system enables interaction with diverse external resources.

Future work could involve enhancing the task planner with more sophisticated reasoning capabilities, expanding the tool library (e.g., web browsing, document interaction), refining the multi-agent collaboration strategies within the `HybridAgent`, and developing more advanced memory retrieval mechanisms (e.g., vector-based similarity search in LTM). Evaluating the framework's performance on standardized agent benchmarks would also be valuable.

7 References

References

[1] Li et al., "From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review"

[2] Zhao et al., "A Survey on Large Language Model based Autonomous Agents"

[3] Wang et al., "The Rise and Potential of Large Language Model Based Agents: A Survey"

[4] Zhou et al., "Large Language Model Agent: A Survey on Methodology, Applications, and Challenges"

[5] LangChain Introduction Documentation, `https://python.langchain.com/docs/introduction/`

[6] Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation", arXiv:2308.08155

[7] Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models", arXiv:2210.03629