The Journal of Open Source Software

OpenAgent: A Modular Framework for Autonomous Multi-Tool Agent Orchestration with Memory-Enabled Planning

Xinyu Zhang [1¶]

1 Independent Researcher ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 19 January 2026
Published: unpublished

Summary

Large Language Models (LLMs) have demonstrated strong capabilities in natural language understanding and generation, yet turning these models into autonomous systems that can reliably execute complex, multi-step tasks remains challenging (Wei et al., 2022; Yao et al., 2023). OpenAgent is a modular, extensible Python framework for building autonomous agents that decompose user requests into plans, orchestrate multiple tools, and preserve intermediate artifacts and context across steps.

The framework provides a hierarchical agent architecture where specialized agents (e.g., planning-oriented, ReAct-style, and software-engineering focused) share a common base interface while implementing distinct reasoning and execution strategies. A key contribution is memory-enabled multi-step execution: OpenAgent tracks intermediate artifacts and summaries to support stateful workflows where later steps can explicitly build on earlier results, enabling tasks such as research-driven document generation and iterative code changes.

Statement of Need

Existing LLM agent frameworks often require significant boilerplate code and lack principled abstractions for multi-tool orchestration. Researchers and practitioners building AI assistants face challenges in:

1. **Task Decomposition**: Automatically breaking complex requests into manageable sub-tasks
2. **Tool Selection**: Dynamically choosing appropriate tools based on task requirements
3. **Context Persistence**: Maintaining relevant context across multi-step executions
4. **Artifact Management**: Tracking and utilizing outputs from intermediate steps

OpenAgent addresses these needs through its layered architecture: a **Tool Registry** pattern enables dynamic tool discovery and registration; **Flow Orchestration** manages execution pipelines; and **Agent Specialization** allows different reasoning strategies (ReAct, hierarchical planning) to be applied based on task characteristics.

The framework targets AI researchers studying agent architectures, developers building task automation systems, and organizations requiring document generation pipelines with integrated web research capabilities.

State of the Field

Recent work has explored tool-augmented LLMs and agentic workflows, including tool-use training and tool learning (Qin et al., 2023; Schick et al., 2023), surveys of LLM agents (Wang et al., 2023; Xi et al., 2023), and composable application frameworks (Chase, 2022). Community systems such as Auto-GPT (Significant Gravitas, 2023), CAMEL (Li et al., 2023), and MetaGPT (Hong et al., 2024) popularized autonomous and multi-agent task execution but often couple planning, tool selection, and execution logic in ways that are difficult to adapt for controlled experiments or specialized domains.

OpenAgent focuses on modularity and experimentation: it separates (i) agent reasoning strategies, (ii) tool definitions and execution, and (iii) flow orchestration into clear abstractions. This separation supports comparative evaluation of reasoning paradigms (e.g., planning vs. ReAct) within a consistent tool/runtime environment, and it enables workflows that require explicit artifact persistence across steps.

Feature	LangChain agents (Chase, 2022)	Auto-GPT (Significant Gravitas, 2023)	OpenAgent
Memory-enabled multi-step execution	Limited	Yes	**Yes**
Modular tool registry	Yes	Limited	**Yes**
Multi-agent specialization	Limited	No	**Yes**
Artifact tracking	No	Partial	**Yes**
Graceful fallback on tool failure	No	No	**Yes**
ReAct reasoning traces	Via LCEL	No	**Yes**
Software-engineering workflow support	No	No	**Yes**

Software Design

OpenAgent implements a three-tier architecture as illustrated in Figure 1:

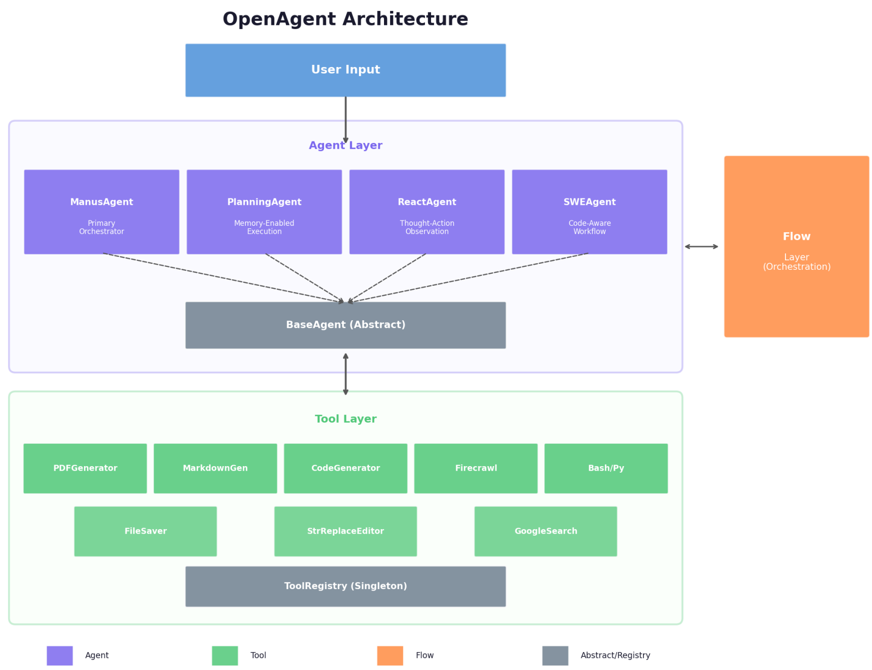

Figure 1: OpenAgent Architecture

Agent Layer

The agent hierarchy implements the Strategy pattern, allowing runtime selection of reasoning approaches:

- **ManusAgent**: Primary orchestrator using LangChain's OpenAI Functions Agent for flexible tool invocation with automatic task-type inference and plan generation
- **PlanningAgent**: Implements memory-enabled multi-step execution with inter-step context passing via `artifact_memory` dictionary, enabling coherent multi-document workflows
- **ReactAgent**: Implements the Reasoning-and-Acting paradigm (Yao et al., 2023) with explicit Thought-Action-Observation cycles, tool selection with fuzzy matching, and configurable iteration limits
- **SWEAgent**: Software engineering specialist following a structured workflow (UNDERSTAND → PLAN → IMPLEMENT → VERIFY → ITERATE) with automatic code verification and iterative bug fixing

Tool Layer

The `BaseTool` abstraction provides: - Standardized parameter schemas using JSON Schema - Automatic error handling and logging - LangChain-compatible `safe_run` interface for heterogeneous invocation patterns - OpenAI function-calling format conversion

Currently integrated tools include: - **PDFGeneratorTool**: Structured document generation with ReportLab, supporting tables, visualizations, and artifact management - **MarkdownGeneratorTool**: Research-oriented document creation with automatic file persistence - **CodeGeneratorTool**: Multi-language code synthesis (Python, JavaScript, Go, Rust, etc.) with optional execution and output capture - **FirecrawlResearchTool**: Web research via the Firecrawl API with data extraction and LLM-based fallback - **BashTool/PythonExecuteTool**: Shell and Python code execution with timeout handling - **StrReplaceEditorTool**: Text and code file editing via search-and-replace operations

Flow Layer

Flows compose agents and tools into reusable pipelines. The `PlanningFlow` demonstrates sophisticated orchestration:

Rather than running as isolated, stateless calls, flows can pass structured task descriptions, execution state, and stored artifacts through multi-step pipelines, enabling reproducible runs of complex tasks (e.g., research → outline → draft → export).

Key Technical Contributions

Memory-Enabled Multi-Step Execution

Unlike stateless agent invocations, OpenAgent's planning system maintains an `artifact_memory` dictionary that accumulates context across steps:

Subsequent steps receive this accumulated context, enabling coherent multi-document workflows where later steps can reference and build upon earlier results.

Dynamic Tool Selection with Fallback

The framework implements graceful degradation: when tool execution fails, an LLM-based fallback generates synthetic results, ensuring workflow completion even under partial failures.

Task-Type Inference

An initial LLM call classifies user input as either conversational or task-oriented, routing requests to appropriate handling paths:

This routing supports lightweight conversational responses as well as multi-step execution when the input is task-like.

ReAct Reasoning Implementation

The ReactAgent implements the ReAct paradigm (Yao et al., 2023) with explicit reasoning traces:

This approach provides transparency through the complete reasoning trace, fuzzy tool name matching for robustness, and configurable iteration limits to prevent infinite loops.

Software Engineering Workflow

The SWEAgent implements a structured five-phase workflow optimized for code-related tasks:

1. **UNDERSTAND**: LLM-based task analysis extracting language, requirements, and complexity
2. **PLAN**: Automatic generation of numbered implementation steps
3. **IMPLEMENT**: Tool-assisted code generation with per-step tool selection
4. **VERIFY**: Automatic execution and testing of generated code
5. **ITERATE**: Error-driven refinement using LLM debugging prompts

Research Impact Statement

OpenAgent is intended to support research and development on autonomous LLM agent behavior by providing a reusable experimental substrate: researchers can compare different reasoning strategies (e.g., planning-oriented vs. ReAct-style execution) while keeping tool interfaces and orchestration constant. For practitioners, the artifact-persistent planning flow enables end-to-end pipelines (e.g., web research and extraction → drafting → PDF/Markdown generation) where intermediate results can be inspected, reproduced, and reused.

The repository includes runnable flows and automated tests that exercise core components (agents, schemas, and tools), supporting reproducibility and regression prevention as the framework evolves.

AI usage disclosure

Generative AI tools were used to edit and reformat portions of this manuscript for clarity and to align with JOSS paper structure. The author reviewed and validated the final wording and all technical claims.

Acknowledgements

We thank the LangChain and OpenAI communities for foundational libraries, and the Firecrawl team for the web research API.

References

Chase, H. (2022). *LangChain: Building applications with LLMs through composability*.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., & others. (2024). MetaGPT: Meta programming for a multi-agent collaborative framework. *International Conference on Learning Representations*.

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., & Ghanem, B. (2023). CAMEL: Communicative agents for "mind" exploration of large language model society. *Advances in Neural Information Processing Systems*.

Qin, Y., Hu, S., Lin, Y., Chen, W., Ding, N., Cui, G., Zeng, Z., Huang, Y., Xiao, C., Han, C., & others. (2023). Tool learning with foundation models. *arXiv Preprint arXiv:2304.08354*.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*.

Significant Gravitas. (2023). *Auto-GPT: An autonomous GPT-4 experiment*.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., & others. (2023). A survey on large language model based autonomous agents. *arXiv Preprint arXiv:2308.11432*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, *35*, 24824–24837.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., & others. (2023). The rise and potential of large language model based agents: A survey. *arXiv Preprint arXiv:2309.07864*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *International Conference on Learning Representations*.