The OpenHands Software Agent SDK: A Composable and Extensible Foundation for Production Agents

Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, Valerie Chen, Robert Brennan, Graham Neubig

Presenters: Jiazhen Liu, Alan Liu
Feb 23, 2026

Background: OpenHands V0

- “OpenHands: An Open Platform For AI Software Developers as Generalist Agents”, Published in ICLR 2025 [1]
- An open platform to advance the development of AI agents
- Towards software engineering generalist
- Requires seamless integration of multiple functionalities

OpenHands V0 becomes popular

About

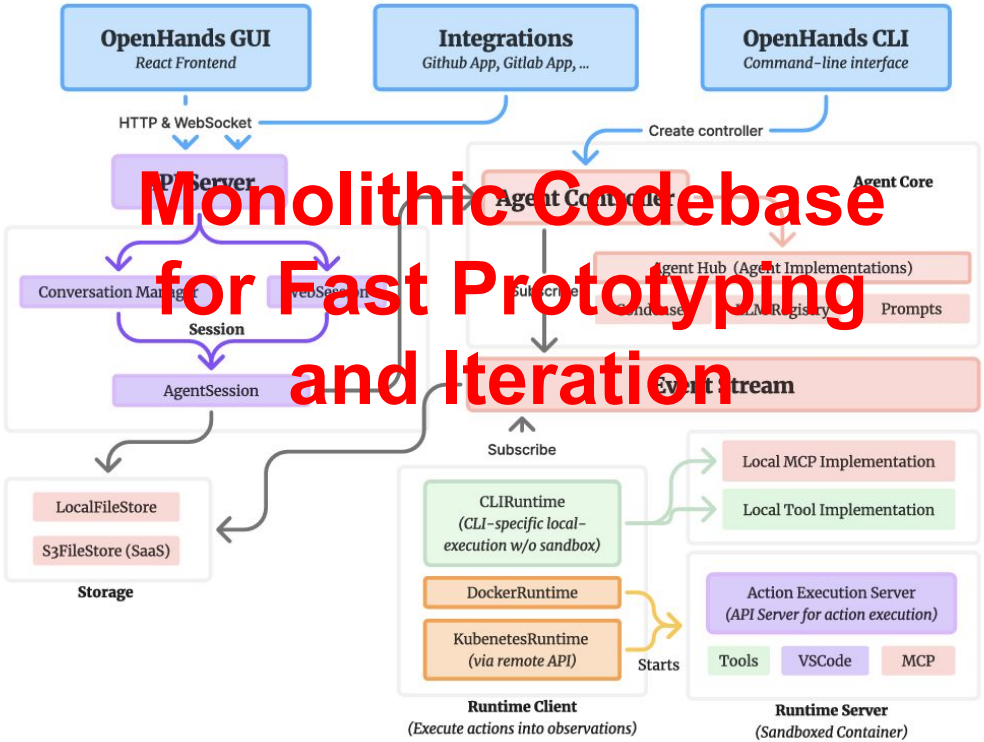
🙌 OpenHands: AI-Driven Development

openhands.dev

agent cli artificial-intelligence
openai developer-tools gpt llm
chatgpt claude-ai

Readme
View license
Code of conduct
Contributing
Cite this repository
Activity
Custom properties
68.1k stars
426 watching
8.5k forks
Report repository

OpenHands V0 faces bottlenecks



Monolithic Codebase for Fast Prototyping and Iteration

Architectural tensions exposed:

❌ Sandbox mandatory

❌ Sprawling mutable configurations

❌ Repeated implementations

Challenges and New Design Principles – I

V0 Assumption: All tool calls should run inside sandboxed Docker containers for safety and reproducibility

- Agent and sandbox run in different processes with potentially divergent states
- Session corrupts when one crashes while the other continues
- One agent’s action crashes other agents in multi-tenant setting

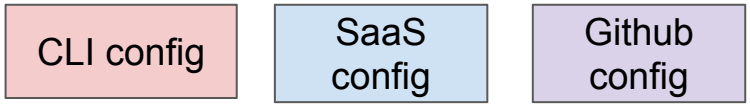
Challenges and New Design Principles – I

V1 Principle: Sandboxing should be opt-in, not universal

- Agent and sandbox run in the same process by default
- Isolation happens when needed
- Improves flexibility without sacrificing safety

Challenges and New Design Principles – II

Non-centralized configurations



- Parallel hierarchies from multiple entry points
- Fixes and patches only take effect locally
- Different dependencies and logic
- Nondeterministic: same parameters ≠ same output
- Small changes → cascading failures

Challenges and New Design Principles – II

V1 Principle: Stateless by default, one source of truth for state

Enables deterministic replay, strong consistency, stable long-term recovery

Challenges and New Design Principles – III

V0 Challenge: Agent core, evaluation suite, and applications all implemented within a monolithic repository

- Blurred the boundary of core agent logic and non-core parts
- Agent absorbs downstream hacks and branches over time
- Messy for maintaining and evolving the system

Challenges and New Design Principles – III

V1 Principle: Maintain strict separation of agent core and applications

- Agent: encapsulated into the software engineering SDK
- Applications: utilize SDK APIs

Challenges and New Design Principles – IV

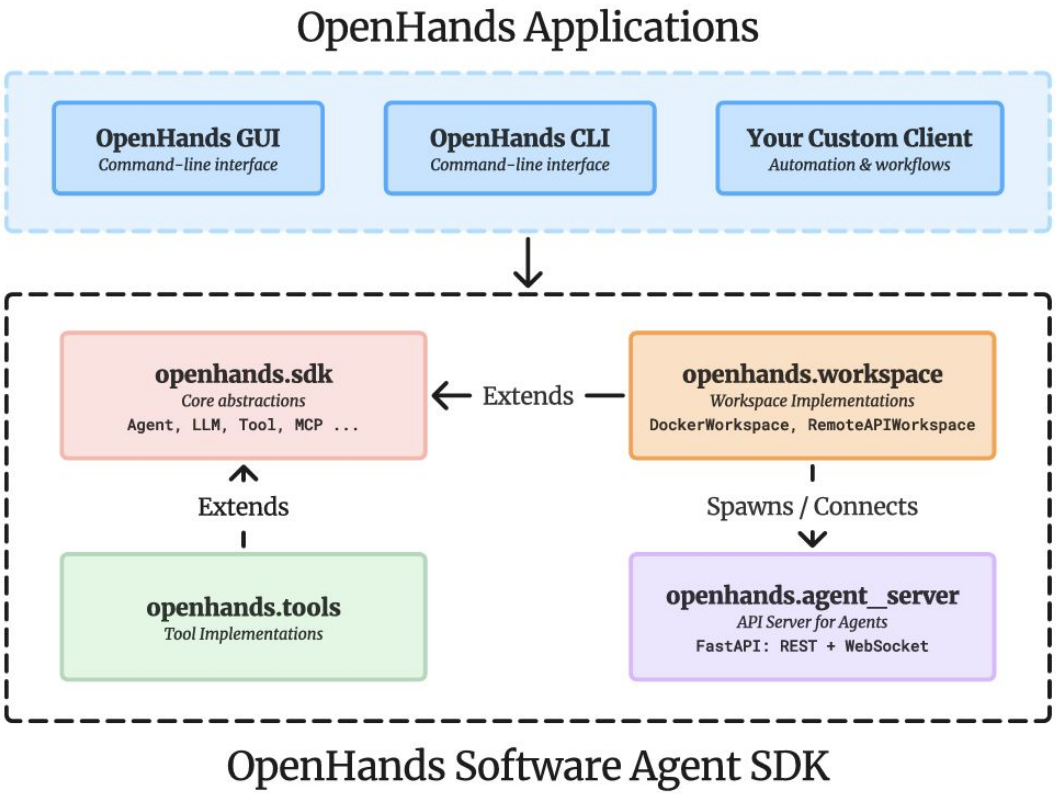
V0 Challenge: No structured notion of composability and extensibility

Even small features requires modifying/branching the agent core!

V1 Principle: Making composability explicit

- Modular design: core, vs everything else
- Extensions don’t touch the core

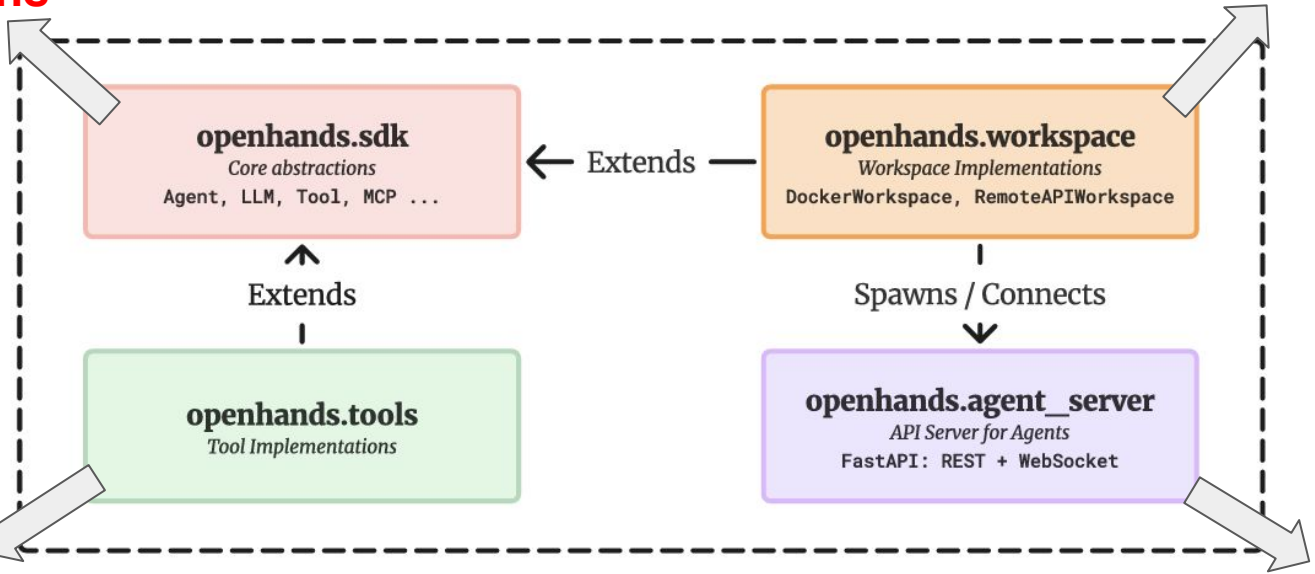
Revamped Architecture in V1



Modularised Design

Lightweight abstractions

Optional sandboxing



Isolated implementations

Interact with server

OpenHands V1 SDK

V1 SDK consists of the following interlocking components:

- Event-sourced state management
- LLM
- Tool system
- Agent (Stateless event processor)
- Context window management
- Local conversation
- Secret registry
- Security and confirmation
- Local & remote workspace support

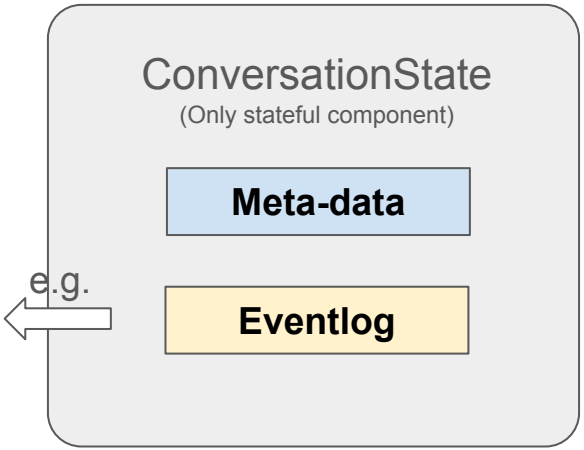
Event-sourced State: Deterministic, Recoverable Agents

LLM, Agent, Tools stay
immutable & serializable

Append-only event log

User Msg → Event
LLM Msg → Event
Tool Call → Event
Tool Output → Event
Condensation → Event
Pause / Resume → Event

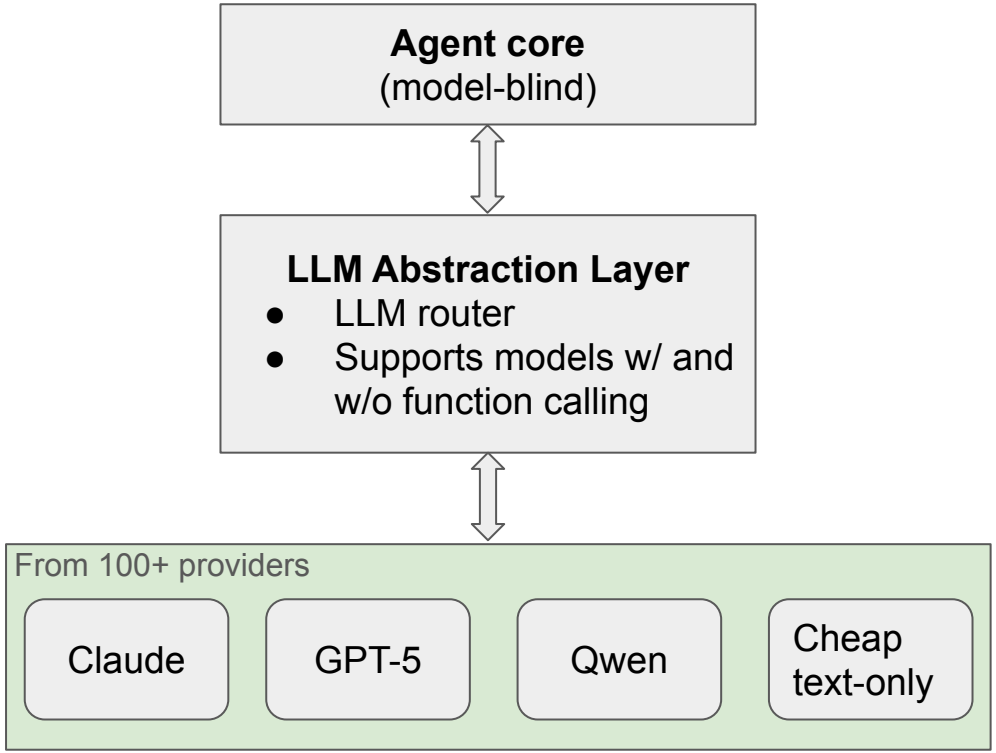
e.g.



Why beneficial?

✅ Deterministic replay
✅ Crash recovery
✅ Debuggable agent
✅ Smooth pause/resume
✅ Context condensation

LLM Abstraction




```python
class RouterLLM(LLM):
    llms_for_routing: dict[str, LLM]  # Available models

    @abstractmethod
    def select_llm(self, messages: list[Message]) -> str:
        """Return key of LLM to use from llms_for_routing."""

    def completion(
        self, messages: list[Message], ... **kwargs,
    ) -> LLMResponse:
        # Select appropriate LLM
        selected_model = self.select_llm(messages)
        self.active_llm = self.llms_for_routing[selected_model]
        return self.active_llm.completion(...)

class MultimodalRouter(RouterLLM):
    def select_llm(self, messages: list[Message]) -> str:
        has_images = any(m.contains_image for m in messages)
        return "primary" if has_images else "secondary"

# Usage: route text to cheaper model, images to multimodal model
router = MultimodalRouter(llms_for_routing={
    "primary": LLM(model="claude-sonnet-4-5"),
    "secondary": LLM(model="devstral-small")})
agent = Agent(llm=router, tools=tools)
```

Customizable LLM Router

Tool System

The tool system uses a three step process:

- Action
- Execution
- Observation

```python
class Schema(BaseModel):
    @classmethod
    def to_mcp_schema(cls) -> dict[str, Any]
    @classmethod
    def from_mcp_schema(cls: type[S], model_name: str, schema: dict[str, Any]) -> type["S"]

class Action(Schema):
    """Tool input with schema validation."""
    def visualize(self) -> str  # For UI display

class Observation(Schema):
    """Tool output with LLM conversion."""
    def to_llm_content(self) -> str  # For LLM context

class ToolDefinition[ActionT, ObservationT](BaseModel):
    action_type: type[Action]
    observation_type: type[Observation]
    executor: ToolExecutor
    def to_mcp_tool(self, ...) -> dict
    def to_openai_tool(self) -> dict  # ChatCompletions API format
    def to_responses_tool(self) -> dict  # Responses API format

class ToolExecutor[ActionT, ObservationT](ABC):
    """Executor function type for a Tool."""
    def __call__(self, action: ActionT) -> ObservationT
```

Tool System

The tool system uses a three step process:

- Action
- Execution
- Observation

```python
class Schema(BaseModel):
    @classmethod
    def to_mcp_schema(cls) -> dict[str, Any]
    @classmethod
    def from_mcp_schema(cls: type[S], model_name: str, schema: dict[str, Any]) -> type["S"]

class Action(Schema):
    """Tool input with schema validation."""
    def visualize(self) -> str  # For UI display

class Observation(Schema):
    """Tool output with LLM conversion."""
    def to_llm_content(self) -> str  # For LLM context

class ToolDefinition[ActionT, ObservationT](BaseModel):
    action_type: type[Action]
    observation_type: type[Observation]
    executor: ToolExecutor
    def to_mcp_tool(self, ...) -> dict
    def to_openai_tool(self) -> dict  # ChatCompletions API format
    def to_responses_tool(self) -> dict  # Responses API format

class ToolExecutor[ActionT, ObservationT](ABC):
    """Executor function type for a Tool."""
    def __call__(self, action: ActionT) -> ObservationT
```

Tool System

The tool system uses a three step process:

- Action
- Execution
- Observation

```python
class Schema(BaseModel):
    @classmethod
    def to_mcp_schema(cls) -> dict[str, Any]
    @classmethod
    def from_mcp_schema(cls: type[S], model_name: str, schema: dict[str, Any]) -> type["S"]

class Action(Schema):
    """Tool input with schema validation."""
    def visualize(self) -> str  # For UI display

class Observation(Schema):
    """Tool output with LLM conversion."""
    def to_llm_content(self) -> str  # For LLM context

class ToolDefinition[ActionT, ObservationT](BaseModel):
    action_type: type[Action]
    observation_type: type[Observation]
    executor: ToolExecutor
    def to_mcp_tool(self, ...) -> dict
    def to_openai_tool(self) -> dict  # ChatCompletions API format
    def to_responses_tool(self) -> dict  # Responses API format

class ToolExecutor[ActionT, ObservationT](ABC):
    """Executor function type for a Tool."""
    def __call__(self, action: ActionT) -> ObservationT
```

Tool System - Why

- The tool system is the execution backbone of the agent
- OpenHands can treat these tools as clean separate components, enabling distributed execution

Stateless Event Processor

What does it mean for the event processor to be “stateless”

- The LLMs processes events and outputs deterministically and solely based on explicit inputs

Why use stateless?

- Secure
- Distributed
- Fault-tolerant

Context, Conversation, and Registry

Context window

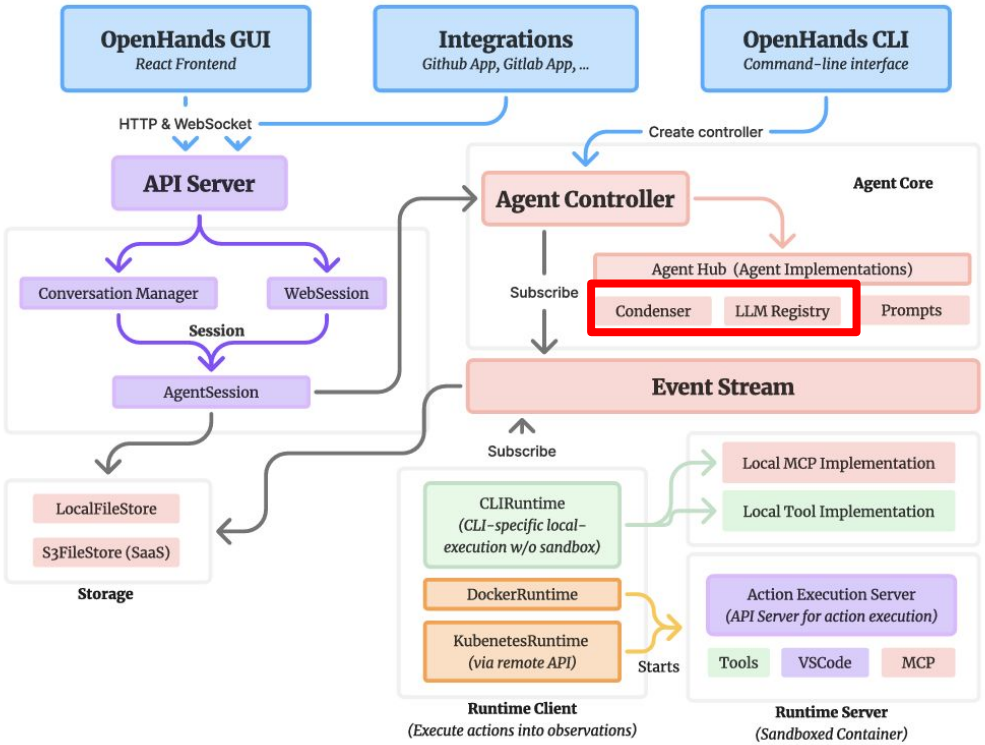
- What the model sees
- Fed by the condenser

Local Conversation

- Execution mode of the SDK
- Runs the agent loop

Secret Registry

- Ensures sensitive information is not leaked



Security and Confirmation

- If the Secret Registry protects data, Security Analyzer protects executions
- This is done by evaluating the risk of a given tool call

Transition from Local to Remote

Key idea: OpenHands can transition from local to remote with minimal code changes

- Local conversations are executed in process
- Remote conversations are delegated to an agent server hosted on a WebSocket

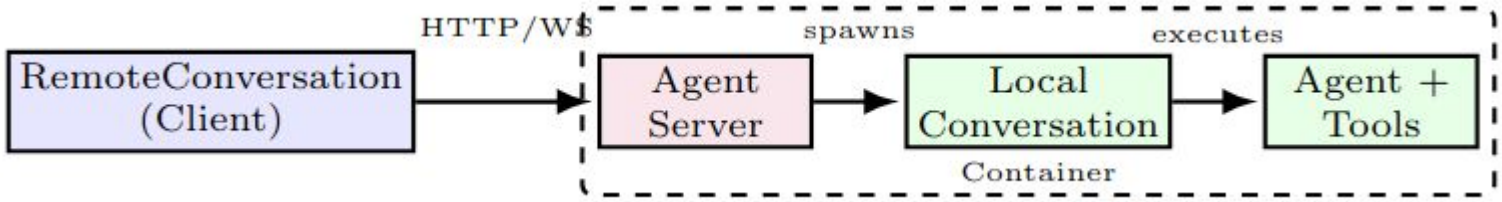
Transition from Local to Remote

```
--- local.py
+++ remote.py
@@
 from openhands.sdk import LLM, Conversation
 from openhands.sdk.preset.default import get_default_agent
 llm = LLM(model="anthropic/claude-sonnet-4.1", ...)
 agent = get_default_agent(llm=llm)
-conversation = Conversation(agent=agent)
-conversation.send_message("Create hello.py")
-conversation.run()
+from openhands.workspace import DockerWorkspace
+with DockerWorkspace(...) as workspace:
+  conversation = Conversation(agent=agent, workspace=workspace)
+  conversation.send_message("Create hello.py")
+  conversation.run()
```

With OpenHands, the only change necessary in the code is instantiating a Docker Workspace

Agent Server

- Uses HTTP commands (GET and POST) to stream events
- Agent is reconstructed on the server, which runs it own Local Conversation
- Container handles execution
- Environment-agnostic



Evaluation - Integration

Benchmark	Model	Performance
SWE-Bench (Jimenez et al., 2024)	Claude Sonnet 4.5	72.8%
	Claude Sonnet 4	68.0%
	GPT-5 (reasoning=high)	68.8%
	Qwen3 Coder 480B A35B	65.2%
GAIA (Mialon et al., 2023a)	Claude Sonnet 4	57.6%
	Claude Sonnet 4.5	67.9%
	GPT-5 (reasoning=high)	62.4%
	Qwen3 Coder 480B A35B	41.2%

Evaluation - SDK Comparison

	OpenAI Agents SDK	Claude Agent SDK	Google Agent Development Kit	OpenHands Software Agent SDK
Standalone SDK Features				
MCP Support	✓	✓	✓	✓
Custom Tools	✓	✓	✓	✓
History Persistence & Restore	✓	✓	✓	✓
Sub-agent Delegation	✓	✓	✓	✓
Model Agnostic (100+ LLMs)	✓	×	✓	✓
Multi-LLM Routing	×	×	×	✓
Conversation Cost & Token Tracking	✓	✓	×	✓
Pause/Resume Agent Execution	✓	×	×	✓
Native Support for Non-function-calling models	×	×	×	✓
Security Analyzer for Agent Action	×	×	✓	✓
Action Confirmation Policies	~	×	~	✓
Context Files (e.g., repo.md, AGENTS.md)	×	✓	×	✓
Agent Skills	×	✓	×	✓
Context Condensation	×	✓	×	✓
TODO List Planner	×	✓	×	✓
Tmux-based Interactive Bash Terminal	×	×	×	✓
Auto-Generated Conversation Titles	×	×	×	✓
Secrets Management with Auto-Masking	×	×	×	✓
Agent Stuck Detection	×	×	×	✓
Long-term Memory across Sessions	×	×	✓	×

Evaluation - SDK Comparison

	OpenAI Agents SDK	Claude Agent SDK	Google Agent Development Kit	OpenHands Software Agent SDK
Production Server Features				
Builtin REST+WebSocket Server	×	×	×	✓
Session-Based Authentication	×	×	×	✓
Builtin Remote Agent Execution	×	×	×	✓
Agent Environment Sandboxing	×	×	~	✓
VNC Desktop for Agent Workspace	×	×	×	✓
VSCode Web for Agent Workspace	×	×	×	✓
Builtin Chromium Browser for Agent	×	×	×	✓

OpenHands is notable for being the first agent SDK to support production server features

Evaluation - SDK Comparison

	OpenAI Agents SDK	Claude Agent SDK	Google Agent Development Kit	OpenHands Software Agent SDK
Continous Quality Assurance				
Programmatic tests	✓	✓	✓	✓
Strong Type Checking	✓	✓	✓	✓
Frequent LLM-based tests	×	✓	✓	✓
Built-in Academic Benchmark Evaluation	×	×	×	✓

Conclusion

- OpenHands uses a stateless, event-sourced, composable architecture spanning four packages (SDK, Tools, Workspace, Server) to bridge the gap between rapid local prototyping and production

References

[1] Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., ... & Neubig, G. (2024). Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

Supplementary slides

Comparison of OpenHands V0 and Other Agents

Framework	Domain	Graphic User Interface	Standardized Tool Library	Built-in Sandbox & Code Execution	Built-in Web Browser	Multi-agent Collaboration	Human-AI Collaboration	AgentHub	Evaluation Framework	Agent QC
AutoGPT Gravitas (2023)	General	✔	✗	✗	✗	✗	✗	✔	✗	✔
LangChain (Chase, 2022)	General	✗	✔	✗*	✗*	✗	✗	✔	✗	✗
MetaGPT (Hong et al., 2023)	General	✗	✔	✗	✔	✔	✗	✔	✗	✔
AutoGen (Wu et al., 2023)	General	✗	✔	✔	✔	✔	✔	✔	✔	✗
AutoAgents (Chen et al., 2024)	General	✗	✗	✗	✗	✔	✗	✗	✗	✗
Agents (Zhou et al., 2023b)	General	✗	✗	✗	✗	✔	✔	✗	✗	✗
Xagents (Team, 2023)	General	✔	✔	✗	✔	✔	✗	✔	✗	✗
OpenAgents (Xie et al., 2023)	General	✔	✗	✔	✔	✗	✗	✔	✗	✗
GPTSwarm (Zhuge et al., 2024)	General	✗	✔	✗	✗	✔	✔	✗	✗	✗
AutoCodeRover (Zhang et al., 2024b)	SWE	✗	✗	✔	✗	✗	✗	✗	✗	✗
SWE-Agent (Yang et al., 2024)	SWE	✗	✗	✔	✗	✗	✗	✗	✗	✗
OpenHands	General	✔	✔	✔	✔	✔	✔	✔	✔	✔

* No native support. Third-party commercial options are available.

This provides more intuition why we need another agent framework