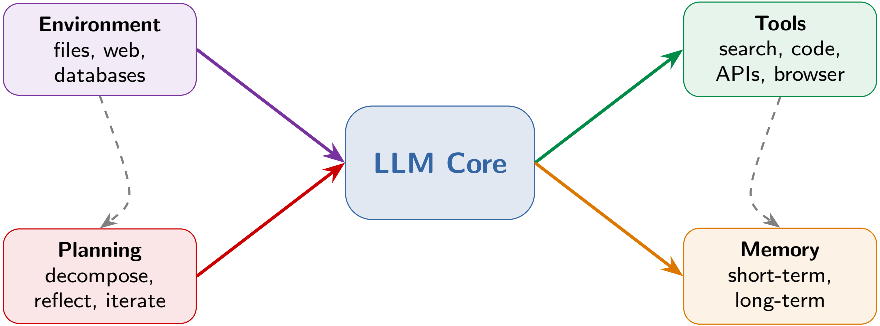
AI Agents & Tool Use

ReAct · Function Calling · Multi-Agent · MCP

What is an AI agent?



An **agent** is an LLM that can autonomously **plan**, **use tools**, and **act** in a loop until a goal is achieved — not just generate a single response.

Chatbot vs. agent

Chatbot

Single prompt → single response

No tool access

No persistent memory

Cannot take actions

"Model as oracle"

→

Agent

Goal → multi-step execution

Tools: search, code, APIs

Short-term + long-term memory

Acts in the real world

"Model as worker"

Andrew Ng's four agentic design patterns (2024):

Reflection — self-critique and improve · **Tool Use** — call external functions

Planning — decompose into subtasks ·
Multi-Agent — specialized roles collaborate

ReAct: Reasoning + Acting

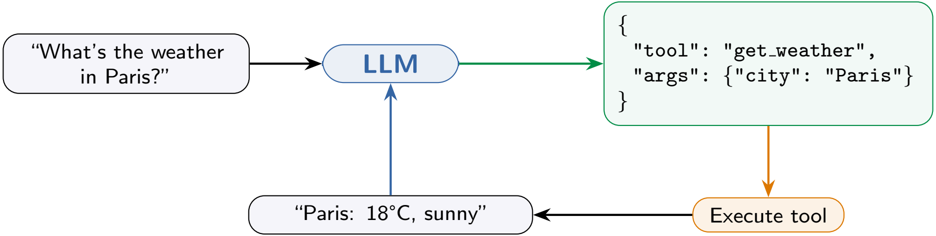
Thought: I need to find when the Eiffel Tower was built.
Action: Search["Eiffel Tower construction date"]
Observation: Construction began in 1887 and was completed in 1889.
Thought: Now I have the answer.
Action: Finish["The Eiffel Tower was built 1887–1889."]

Yao et al., ICLR 2023 (Oral) · ALFWorld: **+34%** over RL baselines

Reasoning helps plan and handle exceptions; actions provide grounded information.

ReAct is now the **default loop** in virtually every agent framework.

Function calling



OpenAI (June 2023): `function_call` API, parallel calls, structured outputs

Anthropic: typed tool schemas, parallel tool use

Google: strict JSON schema enforcement via Vertex AI

Models are **fine-tuned** on (instruction, tool-call, result) triples to learn *when* to call, *which* tool to select, and *how* to format arguments. Structured JSON output is now standard.

Types of tools

Search / Retrieval

Web search APIs
Vector DB (RAG)
Wikipedia, knowledge bases

Code Execution

Python interpreter
Sandboxed runners
Jupyter notebooks

External APIs

Weather, databases
Calendars, e-commerce
CRM, payments

File System

Read / write / edit files
Directory traversal
Document parsing

Browser / Web

Headless browser
Screenshot analysis
Form filling, navigation

Math / Reasoning

Calculator
Symbolic math
Wolfram Alpha

LLMs as Tool Makers (Cai et al., 2023): a powerful LLM *creates* reusable tools (Python functions), then a cheaper LLM *uses* them. GPT-4 maker + GPT-3.5 user $\approx$ GPT-4 quality at lower cost.

Toolformer: self-supervised tool learning

Training text → LLM adds API calls → Keep only if helpful → Fine-tune on result

Original: “The population of Toronto is 2,794,356.”

Annotated: “The population of Toronto is [QA(“population of Toronto”) → 2,794,356].”

Kept because the API call *reduced* perplexity of the next tokens (i.e., helped prediction).

Schick et al., NeurIPS 2023 (Meta) · **6.7B model with tools > GPT-3 175B without**

Tools: calculator, QA, search, translation, calendar

Calculator used in 97.9% of math examples (doubling performance)

Key insight: the model teaches *itself* when and how to use tools, using perplexity as the signal. No human annotation of tool calls needed — fully self-supervised.

Planning & decomposition

HuggingGPT (Shen et al., NeurIPS 2023)

1. Task Planning LLM parses request into subtask list → **2. Model Selection** Pick best model from HuggingFace → **3. Execution** Expert models run each subtask → **4. Response** LLM integrates all results

User: "Describe this image and read any text in it."
Plan: 1. Image captioning (BLIP-2) → 2. OCR (TrOCR) → 3. Combine results

Plan-and-Execute

Create full plan upfront, then execute step by step. Can re-plan if a step fails.

Inner Monologue

Agent maintains a scratchpad of reasoning, tracks progress, and self-corrects along the way.

Memory systems

Short-term

Current conversation in the context window

Limited by context length (4K–200K tokens)

Lost when session ends

Long-term

Persistent storage across sessions via vector DBs

Pinecone, Chroma, Weaviate

Retrieved by relevance

(RAG-style)

Episodic

Memories of past interactions and outcomes

"Last time I tried X, it failed because Y"

Enables learning

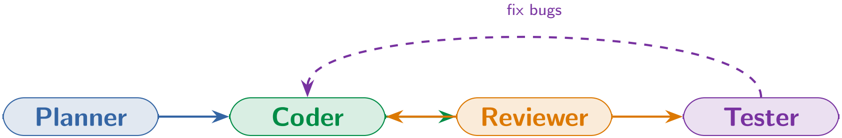
from experience

MemGPT (Packer et al., 2023): OS-inspired virtual memory for LLMs

Main context ("RAM") = limited prompt buffer
$\leftrightarrow$ External storage ("disk") = unbounded archive

Agent manages data movement between tiers using function calls: `search_memory()`, `save_memory()`

Multi-agent systems



AutoGen (Microsoft, 2023)
Customizable conversable agents, flexible topologies
Merged into MS Agent Framework (2025)

CrewAI (2024)
Role-based orchestration
100K+ agent runs/day
60% of Fortune 500

ChatDev (Qian et al., 2023)
Virtual software company: CEO, CTO, programmer, tester, reviewer

Multi-agent debate (Du et al., 2023): multiple LLMs debate and critique each other's answers $\Rightarrow$ improves factual accuracy and reasoning. Diversity of perspectives reduces hallucination.

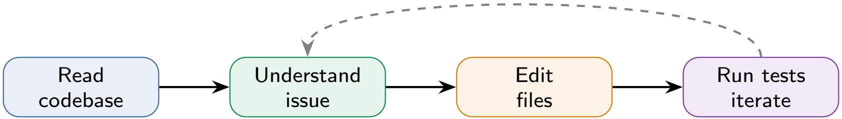
Agent frameworks

Framework	Focus	Key Abstraction
LangChain / LangGraph	General-purpose	Graph-based state machines
LlamaIndex	Data & RAG	300+ data connectors
Semantic Kernel (MS)	Enterprise .NET/Python	Plugins, planners
Claude Agent SDK	Building on Claude	Agent loops, tool use
OpenAI Assistants API	Managed infrastructure	Threads, runs, tools
CrewAI	Multi-agent enterprise	Crews, roles, tasks

Common abstractions: Tools (callable functions) · Agents (LLM + tools + instructions) Memory (short/long-term state) · Chains/Graphs (execution flow) · Planners (decomposition)

Trend: LangChain pivoted from chains to **LangGraph** for complex agent workflows (cycles, state). LlamaIndex dominates RAG-heavy apps. CrewAI dominates multi-agent enterprise.

Code agents



Devin (Cognition, Mar 2024)
"First AI software engineer"
SWE-Bench: 13.9% → GA Dec 2024
ARR: \$1M → \$73M in 9 months

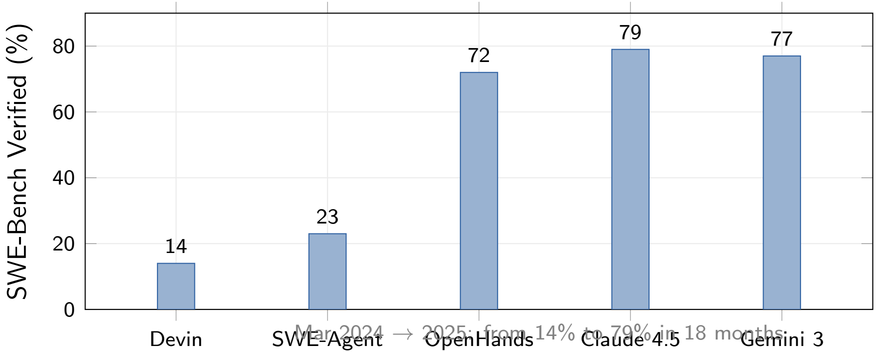
SWE-Agent (Princeton, Apr 2024)
Open-source, custom ACI
Agent-Computer Interface
designed for software engineering

OpenHands (2024)
Open-source platform
SOTA: 72% SWE-Bench Verified
(Claude Sonnet 4.5 + ext. thinking)

Claude Code (Anthropic, 2025)
CLI-based coding agent
Read, edit, run, iterate
Built on Claude Agent SDK

GitHub Copilot Coding Agent (May 2025):
async agent for features, bugs, tests, refactoring.

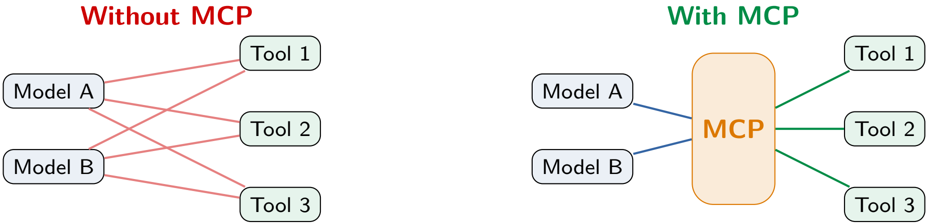
SWE-Bench: the code agent benchmark



Mar 2024 → 2025: from 14% to 79% in 18 months

Jimenez et al., ICLR 2024: real GitHub issues from 12 Python repos

MCP: Model Context Protocol



$N \times M$ integrations

$N + M$ integrations

Anthropic, November 2024 · Like
LSP for AI: standardizes tool integration

JSON-RPC 2.0 over stdio/HTTP · Primitives: **Tools**, **Resources**, **Prompts**

OpenAI adopted (Mar 2025) · Google adopted (Apr 2025) · 97M+ monthly SDK downloads

Challenges

Reliability

95% per-step accuracy
$\Rightarrow$ 60% for 10 steps

One bad step derails the entire workflow

Current: $\sim$80%

Need: $\sim$99%

Hallucinated calls

Non-existent tools
Wrong parameter values
Incorrect types

Hard to catch semantically

(syntax may be valid)

Cost & latency

Complex task = 10–50+ API calls

Cost: 10–100$\times$ a single-shot response

End-to-end: minutes to hours

Safety: autonomous agents act in the real world — editing files, sending emails, making purchases, browsing the web. Unintended consequences are a genuine risk.

Evaluation is hard: multi-step workflows are non-deterministic, have multiple valid paths, and success depends on the *process*, not just the final answer.

Agent benchmarks

Benchmark	What it measures	Scale	Key number
SWE-Bench	Real GitHub issues	2.3K tasks	Best: 79%
AgentBench	8 environments (OS, DB, ...)	29 LLMs	Large open/closed gap
WebArena	Web browsing tasks	812 tasks	4 realistic domains
GAIA	General AI assistant	466 Q	Human: 92%, Best: 65%
ToolBench	API tool use	16K APIs	49 categories

GAIA gap: humans score 92%, best agents score 65%.
Unlike text benchmarks (which are near-saturated), agent benchmarks reveal a **large gap** between human and AI capability on real-world tasks.

Computer use agents

See screen (screenshot) → Reason about what to do → Mouse click or type → Observe result

Claude Computer Use
(Anthropic, Oct 2024)

OSWorld:
14.9% → 72.5%
(∼5× in 16 months)

Operator / CUA
(OpenAI, Jan 2025)

GPT-4o vision + RL
Raw pixels +
virtual mouse

Project Mariner
(Google, Dec 2024)

GUI control via
multimodal models

The paradigm shift: from “model as oracle” to “model as worker.”
Give a goal, the agent plans + acts + observes + iterates until done.

Browsers and operating systems are becoming **agent-native** platforms.

Practical guide

Simple Q&A?
→ Standard LLM (no agent needed)

Need external data?
→ RAG or single tool call

Multi-step workflow?
→ ReAct agent with tools + memory

Complex software task?
→ Code agent (SWE-Agent, Claude Code, OpenHands)

Need specialization?
→ Multi-agent system (CrewAI, AutoGen)

Standardize tools?
→ MCP servers (build once, use everywhere)

Rule of thumb: use the simplest approach that works.
Single LLM call → tool call → ReAct loop → multi-agent. Add complexity only when needed.

Further reading

Foundations

- Yao et al. (2022), "ReAct: Synergizing Reasoning and Acting in Language Models"
- Schick et al. (2023), "Toolformer: Language Models Can Teach Themselves to Use Tools"
- Shen et al. (2023), "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends

Multi-Agent & Code Agents

- Wu et al. (2023), "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation"
- Yang et al. (2024), "SWE-agent: Agent-Computer Interfaces Enable Automated

Surveys

- Wang et al. (2024), "A Survey on Large Language Model based Autonomous Agents"
- Xi et al. (2023), "The Rise and Potential of Large Language Model Based Agents: A Survey"

Questions?

All DL4NLP topics complete!