How to build an AI agent

Building AI agents, explained

An AI agent is a system that can leverage external tools and data sources to execute tasks with minimal human intervention. Though AI agents can also be powered by reinforcement learning (RL) or even explicit rules, in modern parlance the term usually refers to a system with a large language model (LLM) as its engine. While there are many frameworks and platforms for building AI agents with LLMs, catering to nearly any level technical expertise, they all rely on in-depth understanding of your agent's users and purpose to succeed.

LLM-driven agents can not only understand the intent of user inputs, but also discern actions to be taken in response and generate outputs that make them happen.

On the most fundamental level, a generative AI agent is just an LLM operating in an environment in which its natural language processing (NLP) ability is used to output text that can function as the input to external tools and data sources to which it has been connected. For example, the LLM driving an agent might output a SQL query to extract information from a company database, or a structured JSON object to make an API call that triggers a specific action in an external application.

Those LLM outputs are not actions themselves: they simply *trigger* actions upon being sent to those external tools. If the LLM is provided a clear set of logic regarding which actions it should trigger in response to certain feedback from its environment, it can autonomously plan and perform tasks in a manner more dynamic than older, explicit rules-based systems like robotic process automation (RPA).

One way to set up that agentic architecture—to connect an LLM to the APIs for external tool calls and data sources, establish the logic for what outputs it should generate and when, and so on—is to code it from scratch in any common programming language, such as Python or Javascript. This allows for the greatest degree of control, customization and transparency, but also requires the most manual work and technical knowledge.

As an alternative, there exist many services designed to expedite the process of building and deploying your own AI agents. AI agent frameworks streamline the coding process through predefined building blocks while AI agent platforms provide an end-to-end solution for building and deploying agents. The optimal choice for your first agent will depend on your skill and experience level, as well as the nature of your intended use cases.

In any case and at any skill level, the process of building your own AI agent should begin the same way: by carefully defining your AI agent's purpose and identifying the tools and information needed to achieve that purpose.

Defining your AI agent's purpose

The first and most important step in building an AI agent is understanding your needs, how an AI agent might address them and how you can objectively measure its performance. A simple agent with a clearly defined problem to solve and well-mapped solution to that problem is more likely to succeed than a highly sophisticated agent whose workflow is poorly suited to the task at hand.

The problem your AI agent is intended to solve will dictate both logical and technical decisions in its design. A customer support agent entails entirely different prompts, tools and environments than an automated code review agent or a finance agent designed to query company revenue data and provide business intelligence insights. A narrowly-defined problem might be addressed with a single agent, while a broadly-defined problem might require a multi-agent system.

Breaking down the basics

Start with straightforward questions about your agent.

Who will use it? Technical users and non-technical users have different expectations, from how actively they'll need to manage the workflow to the kind of user interface they'll be comfortable with.

What are its responsibilities? You'll need to decide if the AI agent is intended to tackle the entire problem or simply automate away some of the more tedious subtasks required to solve it.

What kinds of inputs will it process? An agent that will process multiple data modalities—text, audio, image, video—might require multiple models. Multimodal models exist, but sometimes a single AI model won't provide adequate accuracy across all of your specific needs. For instance, if your agent needs to extract text from images and PDFs, an optical character recognition (OCR) model might outperform a generalist vision-language model (VLM). If those PDFs contain complex tables, charts and equations, you might need a dedicated document conversion model. If your agent will be working with very niche or domain-specific examples, its constituent models might need fine-tuning on a custom dataset.

What data sources and knowledge bases will it need access to? One of the most basic and essential agentic AI functions is retrieval augmented generation (RAG). A customer support chatbot, for instance, might need to reference information from your company's CRM platform or FAQs. A software engineering agent will likely need to reference your codebase. Many agents need access to real-time information through search engines or specific web services. Knowing not just the information an AI agent will need, but also where it can be found, is essential.

What tools will it need access to? One of the fundamental benefits of an AI agent is the ability to supplement an LLM's capabilities through external tool use. An agent acting as an "AI assistant" should be able to write and modify calendar items. Coding agents need a sandboxed environment to

execute scripts. An AI agent in a sales or marketing environment might need external apps to send emails and other communications. External calculators or computational engines like Wolfram Alpha ensure mathematical operations can be performed reliably and quickly.

How will you measure success? AI agent development is an iterative process of testing, deploying, evaluating and optimizing. Depending on your use case, "success" might variably be measured as a function of things like time saved, cases reviewed, completion rates, accuracy or direct user ratings.

Practical priorities and constraints

You'll also need to factor in practical considerations.

- Will your AI agent be operating in a time-sensitive environment in which speed is essential? Or will accuracy be more important? Different priorities often call for different models and differing levels of workflow complexity.
- What kind of hardware and budget do you have available? Different LLMs entail different costs and computational requirements, whether through API pricing (for closed-source models) or hardware and cloud computing costs.
- Will the input prompts your AI agent responds to be highly varied and complex? Will they be simple and repetitive?

As is often the case in the world of LLMs, the "best" agent is not necessarily the most sophisticated one: it's the one that most efficiently matches your needs and limitations. Simpler agentic systems are easier and cheaper to operate, easier to debug and easier to explain to the people who need to use them.

Choosing the right model(s)

One of most important decisions to make is which specific model (or models) will serve as the decision-making engine of your autonomous agent.

As a general rule, different priorities call for different LLMs. For some use cases, speed might take precedence over topline accuracy. In other environments, precision is paramount. The former scenario calls for smaller, faster LLMs; the latter benefits from larger, more performant models. When an agentic workflow entails particularly complex details or multi-step planning, it might be worth considering a reasoning model. If your workflow entails straightforward and repetitive tasks, it might not warrant the additional latency and computational costs.

In many cases, your workflow might benefit from a *multi-agent system*. In a typical multi-agent setup, an agent driven by a larger LLM intakes the initial request and breaks it up into subtasks, which it then delegates to agents driven by smaller LLMs that quickly execute specialized tasks.

The same considerations should factor into your choice of other types of models. Does your use case involves a computer vision task such as object detection or image segmentation? If real-time responses are essential, a speedy convolutional neural network (CNN)-based model might be the best fit. If maximum accuracy is most important, a slower but more performant vision transformer (ViT) might be better.

In the context of agentic AI, users should consider different LLM performance metrics than they would prioritize in the context of chatbot-driven applications. For instance, performance on benchmarks that

reflect world knowledge and creative writing might be useful for a chatbot, but be of little importance to an AI agent that will have access to external data sources and output brief, purpose-driven text. For agentic AI, a model's function-calling and instruction-following abilities usually take precedence.

Working with AI agent frameworks and platforms

If you're not prepared to code your AI agent from scratch, you'll want to explore different *AI agent frameworks* or *AI agent platforms*.

AI agent frameworks are, in essence, code libraries that provide modular components designed to be assembled together into agentic workflows. Rather than having to code every single element from scratch, developers can use an agentic framework's pre-defined code snippets, classes and functions as building blocks to simplify and streamline the process. Ideally, frameworks provide a quicker and more scalable approach than raw code while still allowing for a high degree of control and customization. components designed to be assembled together into agentic workflows. Rather than having to code every single element from scratch, developers can use an agentic framework's pre-defined code snippets, classes and functions as building blocks to simplify and streamline the process. Ideally, frameworks provide a quicker and more scalable approach than raw code while still allowing for a high degree of control and customization.

AI agent *platforms*, such as IBM watsonx.Orchestrate, are more comprehensive. They encompass not only the tools to build agents, but also the infrastructure for hosting, deploying, monitoring and managing those agents at scale. Many AI agent platforms provide low-code or even no-code user interfaces (UIs), allowing even beginners and non-technical users to build and use AI agents for real-world applications using a template-based approach.

Frameworks and platforms are often not mutually exclusive. Though some platforms restrict users to no-code workflows or are designed exclusively for compatibility with specific models or frameworks, others—including watsonx.Orchestrate—are framework-agnostic and suited to multiple skill levels. Such platforms usually include a dedicated UI for building agents, but also provide users the option of building agents through a framework of their choosing and simply using the platform to deploy and manage them.

AI agent frameworks

IBM's Guide to AI Agents provides overviews, step-by-step guides and tutorials for a wide variety of open-source frameworks for building AI agents. Here, in alphabetical order, are links to explainers and tutorials for a number of well-known frameworks.

Autogen is an open-source framework maintained by Microsoft designed for scalable multi-agent AI systems. To help get started, you can explore this tutorial for multi-agent RAG with AutoGen.

BabyAGI is an experimental open-source agentic framework built and maintained by Yohei Nakajima.

ChatDev is an open-source agentic framework designed to simulate a virtual software company, in which various AI agents perform different roles within that organizational structure. It's intended to serve as an ideal environment for studying *collective intelligence*, with agents prompting each other to solve specific tasks through role-playing. Learn more through this tutorial for collaborative software development in ChatDev.

crewAI is a prominent Python-based, open-source orchestration framework optimized for multi-agent workflows, configurable for use with any open-source LLM or API. To get familiar with it, check out

this tutorial for retail shelf optimization with multimodal, multi-agent systems.

IBM watsonx Orchestrate offers an array of **watsonx agents** optimized to autonomously perform many commonplace workplace functions.

LangChain is among the earliest and most popular orchestration frameworks for open-source AI agents. Whereas LangChain is optimized for "chaining" together tasks in linear workflows, **LangGraph** is a graph-based framework designed for more complex, non-linear workflows. You can experiment with LangChain through this tool-calling tutorial or try out LangGraph with tutorials for building a SQL agent or an IT support ReAct agent.

LangFlow is low-code and no-code open-source framework built upon a drag-and-drop graphical user interface (GUI).

MetaGPT is a multi-agent framework in which each agent acts an employee following a streamlined and strictly specialized workflow. You can experiment with it using this tutorial for multi-agent automation of product requirement document (PRD) creation.

AI agent platforms

AutoGPT is an open-source platform originally designed for integrating OpenAI's GPT models into agentic workflows. It now also offers integration with a wide variety open models, as well as additional proprietary models such as Claude or Gemini.

BeeAI is the first open-source platform built on the Agent Communication Protocol (ACP), an open standard designed to enable smooth communication between AI agents built on any framework. Originally developed by IBM Research, BeeAI is now hosted on the Linux Foundation. To help get started, you can check out this tutorial for a multi-agent contract management system with BeeAI.

IBM watsonx Orchestrate is a framework agnostic, end-to-end platform for building, deploying and coordinating agents across an enterprise. It offers workflows ranging from drag-and-drop, no code AI agent builder UIs to full pro-code customization. Users can leverage pre-built agents, develop their own agents within the platform or import third party agents from elsewhere as they see fit.

AI agent protocols

As complex multi-agent systems have begun to proliferate across the AI ecosystem, a number of *AI agent protocols* have been proposed to standardize communications between AI agents and the APIs of other systems. Though the industry is expected to eventually rally around a single standard, there are currently multiple protocols in use.

Most frameworks endeavor to accommodate all well-known protocols, but in some cases your preference of protocol might limit your choice of framework and vice versa. Some prominent AI agent protocols include:

- **Agent Communication Protocol (ACP)**: Originally introduced by IBM's BeeAI, ACP aims to define and standardize the means through which AI agents operate and communicate with one another. ACP uses REST-based communication based on standard HTTP conventions, making it easy to integrate AI agents into production. It's designed for asynchronous communication by default and doesn't require any specialized libraries. You can explore it through this tutorial.

- **Agent2Agent (A2A)**: Originally launched by Google and other partners under the Google Cloud Platform in April 2025, it's now hosted by the Linux Foundation as an open-source project. It follows a client-serve model setup with a three-step workflow: discovery, authentication and communication. For hands-on experience with A2A, check out this tutorial. It's worth noting that the Linux Foundation, which now operates both ACP and A2A, is working to merge the two protocols by migrating ACP functionality into A2A.

- **Model Context Protocol (MCP)**: Introduced by Anthropic, MCP standardizes communications through a system of conversions into JSON-RPC format. MCP has been adopted by many prominent technology providers, including Figma, Notion, Atlassian, Zapier, Striper, PayPal and Square. To learn more about MCP, visit this tutorial for building an MCP