READ MODE · ANSWER FIRST

Building AI Agents

An interactive, visual explanation of Building AI Agents, from setup through the complete system.

CHEAT SHEET · 6 ESSENTIAL IDEAS

The whole story in 6 lines

Building AI Agents becomes easier to reason about when every stage is connected as one system.

1. The agent loop is think-decide-act-observe-repeat. Each iteration adds new information.
2. The harness is the orchestration layer. It manages the conversation history, parses tool calls, enforces limits, and decides when to stop.
3. MCP is like USB for AI tools. One protocol, many servers. The agent does not need custom code for each tool.
4. A tool call flows through five hops: LLM emits JSON, harness parses it, MCP client sends the request, MCP server executes the tool, and...
5. A complex query often needs multiple tools called in sequence. The LLM breaks the question apart and plans which tools to call and in...
6. Multi-step execution is the agent loop in action. Each iteration adds evidence.

STAGE 1 OF 8

Setup

Welcome. You are about to learn how to build an AI agent from scratch. Not a chatbot that responds once, but an agent that can think, plan, call external services, and synthesize results. Think of it like building a research assistant that knows how to use a library.

An LLM is a large language model. It is the brain of the agent. It reads text, reasons about it, and produces text back. But on its own, an LLM cannot look anything up or take action in the real world.

An agent is an LLM wrapped in a loop. Instead of responding once, it can think about what to do, take an action, observe the result, and decide whether to keep going or stop. The loop is what makes it an agent.

A harness is the code that runs the agent loop. It sends prompts to the LLM, parses tool call requests from the response, executes those tools, feeds results back, and decides when to stop. The harness is your code. The LLM is the model.

A tool is any external capability the agent can use: searching a database, calling an API, reading a file. MCP, the Model Context Protocol, is an open standard that lets your agent discover and call tools through a uniform interface, regardless of where those tools live.

OpenSearch is a search and analytics engine we will use as our example tool. It can run complex queries over large datasets. Over the next six stages, we will build an agent that connects to OpenSearch and an analytics service to answer complex user questions. Let us start with the most fundamental idea: the agent loop.

STAGE 2 OF 8

Agent Loop

02 AGENT LOOP

Here is the core idea. A chatbot receives a question and responds. An agent receives a question and enters a loop. Each pass through the loop adds new information. Watch how the cycle unfolds.

The user sends a question. The LLM reads it and thinks about what information it needs. This is the "think" phase. The LLM is not answering yet. It is planning its next move.

The LLM decides it needs external data and emits a tool call request. This is the "act" phase. The request travels to a tool, which executes and returns a result. That result is the "observe" phase.

The tool result goes back into the conversation. The LLM now has new evidence. It re-reads everything and decides: do I have enough to answer, or do I need another tool call? Try changing the Iterations control in the sidebar to see more loops.

Switch the Outcome control to Error to see what happens when a tool fails. The LLM sees the error, reasons about it, and can retry with different parameters or try a different approach entirely. The loop makes error recovery natural.

When the LLM decides it has enough information, it produces a final text answer instead of another tool call. The loop exits. The key takeaway: the loop is what gives an agent its power. But who runs this loop? That is what the harness does. Let us look at that next.

STAGE 3 OF 8

The Harness

03 THE HARNESS

The agent loop from the previous stage does not run itself. Your code, the harness, drives every step. Think of the harness as the conductor of an orchestra: the musicians (LLM and tools) play, but the conductor keeps everyone in sync.

The harness starts by building the initial message list: a system prompt that defines the agent's role, the available tools, and the user's question. This message list is the agent's memory.

Each iteration, the harness sends the message list to the LLM and parses the response. If the response contains a tool call, the harness extracts the function name and arguments. If it contains plain text, the loop is done.

The harness executes the tool call and appends both the call and the result to the message list. This growing history is how the agent builds context across iterations. Each loop sees everything that happened before.

A critical job: the harness enforces limits. A max iteration count prevents infinite loops. A token budget prevents runaway costs. Try changing the Max Loops control to see how the harness caps the agent. Without these limits, a confused agent could spin forever.

Switch the Mode control to Batch to see the difference. In streaming mode, the harness processes tokens as they arrive, so you see partial results early. In batch mode, it waits for the complete response. The harness handles both. Now that we know what the harness does, let us look at how it connects to tools.

STAGE 4 OF 8

Tools & MCP

04 TOOLS & MCP

We have the loop and the harness. Now the agent needs hands. Tools are how the agent interacts with the outside world. But wiring up each tool individually would be a nightmare. That is where MCP comes in.

MCP, the Model Context Protocol, is like USB for AI tools. Instead of writing custom integration code for each service, your agent speaks one protocol. Any MCP server can plug in. The agent discovers what tools are available at startup.

Each MCP server wraps one external service. In our example, we have an OpenSearch server that exposes search and aggregation tools, an analytics server for statistical analysis, and optionally a file system server. Try changing the Servers control to add or remove them.

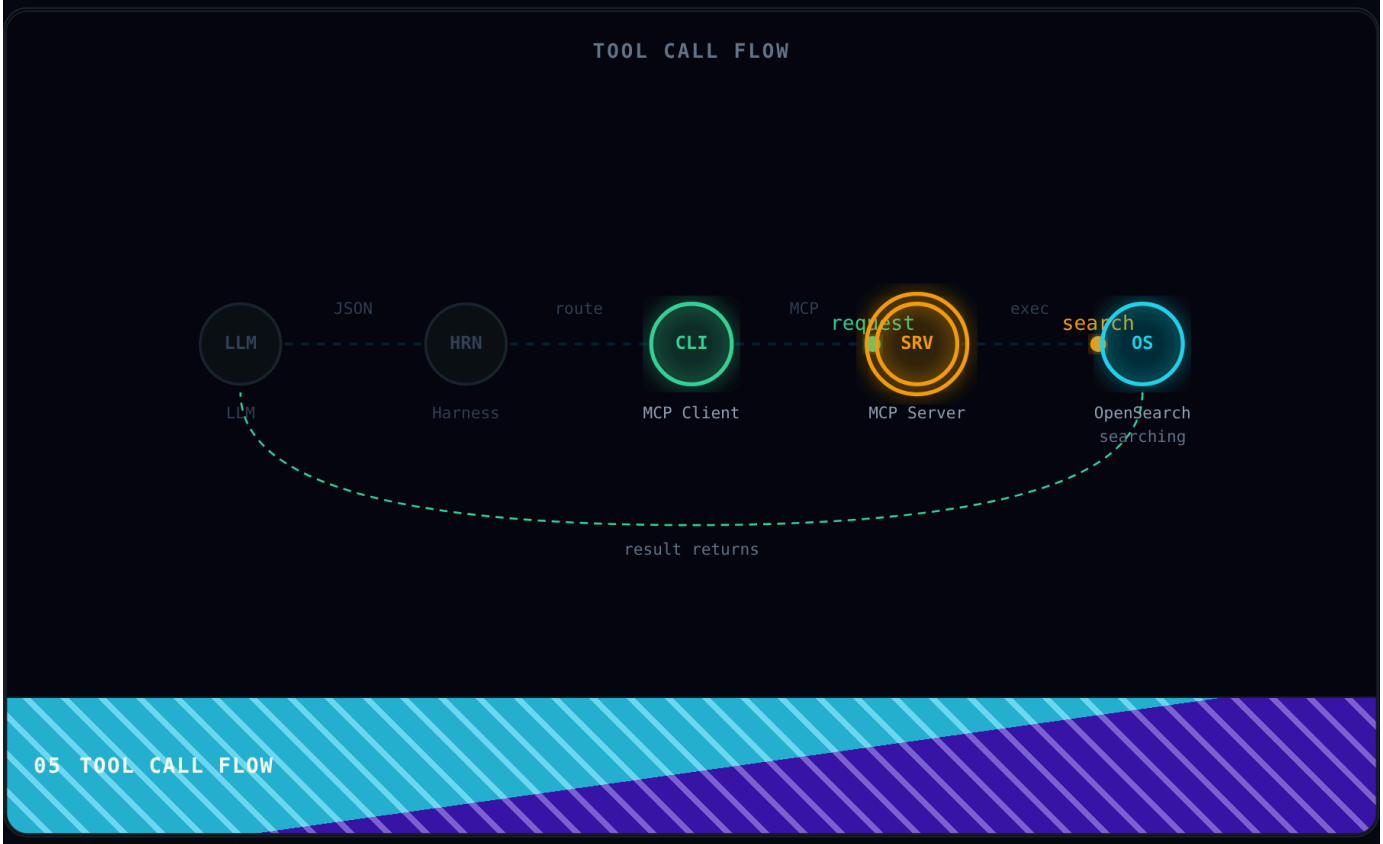
When the agent starts, the MCP client sends an initialize handshake to each server. The server responds with a list of tools it offers: names, descriptions, and parameter schemas. The harness feeds these tool definitions to the LLM so it knows what is available.

Switch the Discovery control to Static to see the difference. With static discovery, the tool list is hardcoded in the harness config. With dynamic discovery via MCP, the agent adapts to whatever servers are connected. Dynamic discovery is more flexible but requires the MCP handshake.

That is the power of MCP: your agent code stays the same regardless of which services it connects to. Add a new MCP server, and the agent automatically learns its tools. Now let us trace exactly what happens when the LLM decides to call one of these tools.

STAGE 5 OF 8

Tool Call Flow



05 TOOL CALL FLOW

Now that we understand MCP from the previous stage, let us trace a single tool call end to end. Think of it like following a letter through the postal system: it passes through several hands before reaching its destination and coming back.

The LLM does not call tools directly. It outputs a structured JSON object that says: "I want to call this tool with these arguments." The harness detects this tool call in the LLM response and extracts the function name and parameters.

The harness routes the call to its MCP client. The client knows which MCP server handles this tool, because it learned that during the discovery handshake we saw in the previous stage. It sends a tools/call request over the MCP protocol.

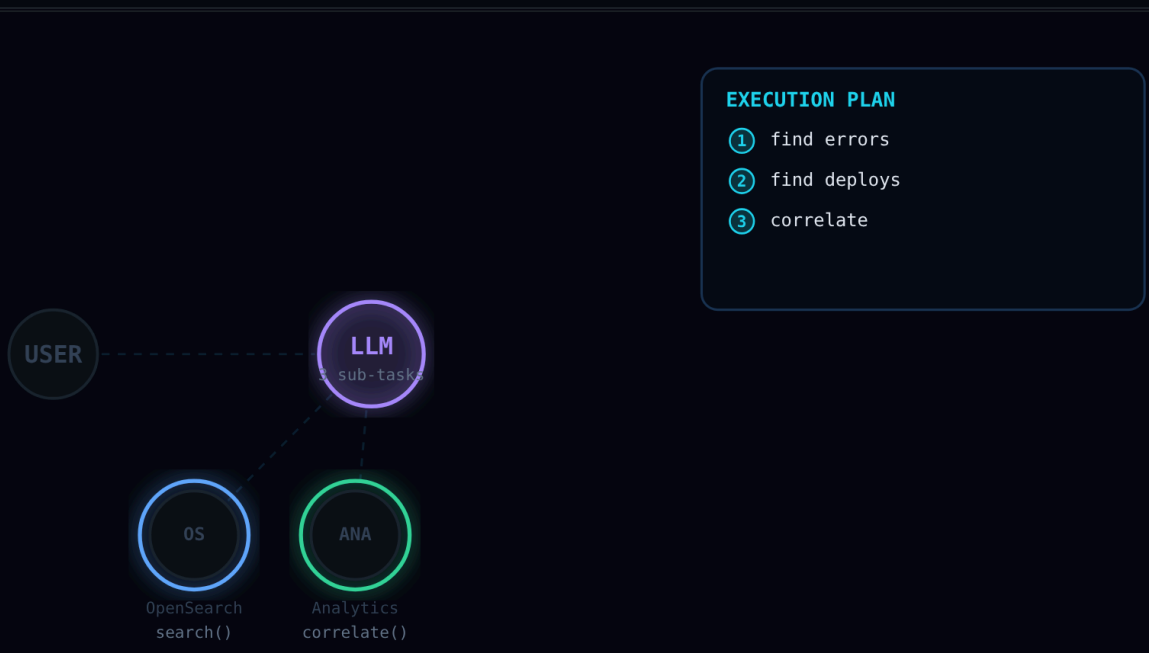
The MCP server receives the request and executes the actual operation. For our OpenSearch example, this might be running a search query or an aggregation. The real work happens here, outside the LLM. Switch the Tool control to Aggregate to see a different operation.

The result travels back: MCP server to MCP client to harness. The harness formats the result as a tool response message and appends it to the conversation history. Switch the Result control to Error to see how failures are handled. The error message goes back to the LLM just like a success.

The LLM now sees the tool result in its conversation history and can reason about it. One full round trip is complete. The key insight: the LLM never touches the external service directly. Every call flows through the harness and MCP. That separation is what makes agents safe and debuggable. Next, let us see this in action with a real query.

STAGE 6 OF 8

Query Planning



06 QUERY PLANNING

Time to see everything come together. A user asks: "What are the top error patterns in the last 24 hours and how do they correlate with deployment events?" This is not a question any single tool can answer.

The LLM reads the question and enters the think phase from our agent loop. It recognizes this needs multiple steps: first find the errors, then find the deployments, then correlate. This decomposition is the agent planning.

The plan maps each sub-task to a specific tool. Step one: call the OpenSearch search tool to find error logs. Step two: call OpenSearch again to find deployment events. Step three: call the analytics service to correlate the two datasets.

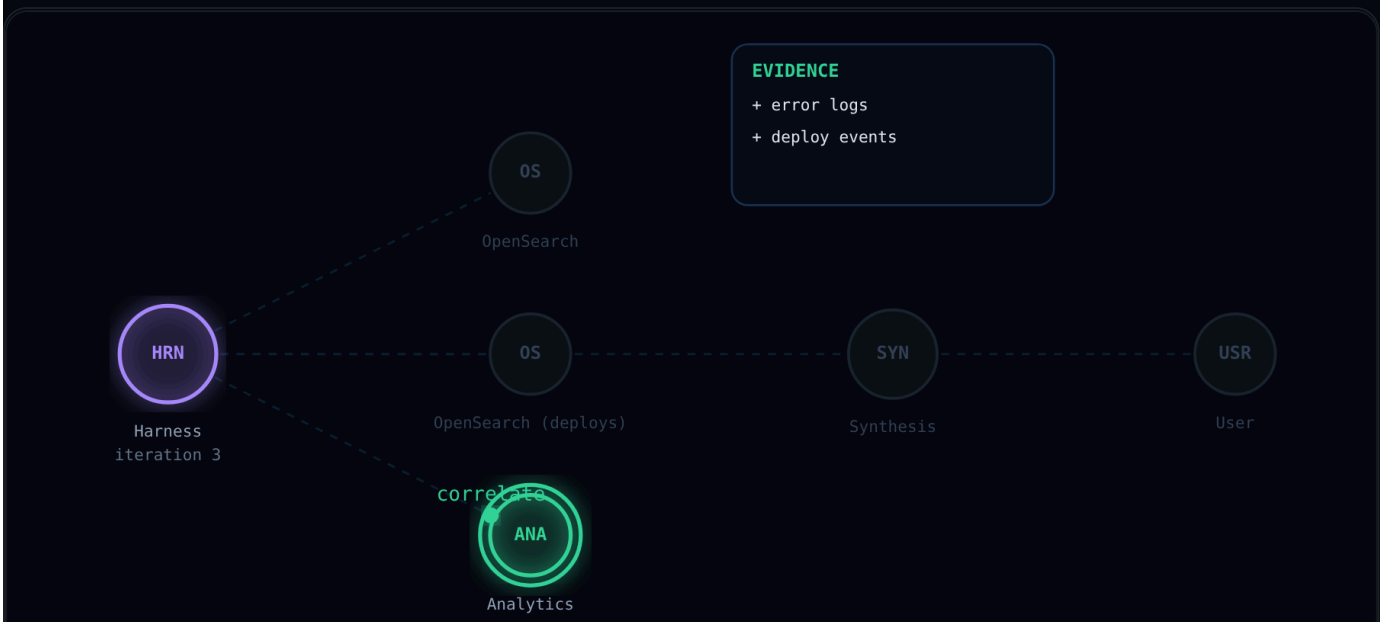
Switch the Complexity control to Simple to see the difference. A simple query like "How many errors in the last hour?" maps to a single aggregation call. Complex queries are where agents shine, because the LLM figures out the multi-step plan automatically.

Change the Services control to see how adding more backend services expands the agent's capabilities. With two services (OpenSearch and analytics), the agent can search and correlate. With three, it might also pull deployment metadata from a separate service.

That is query planning: the LLM decomposes a hard question into a sequence of tool calls. The harness will execute each call through MCP, feed results back, and let the LLM decide the next step. Let us watch that execution unfold.

STAGE 7 OF 8

Execution



07 EXECUTION

The plan from the previous stage is ready. Now the harness starts executing. Remember the agent loop from Stage 1? This is that loop in action, running multiple iterations with real tool calls through MCP.

Iteration one: the LLM asks OpenSearch to search for error logs from the last 24 hours. The call flows through MCP exactly as we traced in Stage 4. The search results come back with timestamps, error types, and counts.

Iteration two: the LLM now knows what errors exist. It asks OpenSearch again, this time for deployment events in the same time window. Notice how the second call is informed by the first result. The LLM uses the error timestamps to narrow the deployment search.

Iteration three: both datasets are in the conversation. The LLM calls the analytics service to correlate errors with deployments. Try changing the Steps control to see how more or fewer iterations change the depth of analysis.

The LLM now has all the evidence. Instead of emitting another tool call, it produces a text response. This is the synthesis step: it reads the full conversation history and writes a unified answer drawing on every result it gathered.

The user receives a complete analysis: "Three error spikes occurred at 14:00, 18:30, and 22:15. The first two correlate with deployments v2.3.1 and v2.3.2. The third appears unrelated." That answer required three tool calls, two services, and five loop iterations. And it all ran through one harness, one protocol, and one loop. Now let us step back and see the whole picture.

STAGE 8 OF 8

Recap

08 RECAP

We started with the agent loop: think, decide, act, observe, repeat. This is the core idea that separates an agent from a chatbot. The loop gives the agent the ability to gather information iteratively until it has a complete answer.

Then we built the harness: the code that runs the loop. It manages the conversation history, parses tool calls from LLM responses, executes them, enforces iteration limits, and decides when to stop. The harness is your code. The LLM is the brain.

We connected tools through MCP: one protocol, many servers. Instead of custom integration code for each service, MCP gives the agent a standard way to discover and call any tool. Like USB for AI.

We traced the full tool call flow: LLM emits JSON, harness parses it, MCP client routes it, MCP server executes it, result flows back. That separation keeps the agent safe, debuggable, and modular.

We saw query planning in action: the LLM decomposed a complex question into sub-tasks, each mapped to a specific tool call. The planning happens inside the LLM. The execution happens through the harness and MCP.

And finally, multi-step execution chained it all together. Three tool calls, two services, five loop iterations, one unified answer. The loop, the harness, MCP, and the tools worked in concert. That is what building an AI agent looks like.