LangChain Docs

Streaming › Overview

Core components › Streaming

Overview

Copy page

Stream real-time updates from agent runs

LangChain implements a streaming system to surface real-time updates.

Streaming is crucial for enhancing the responsiveness of applications built on LLMs. By displaying output progressively, even before a complete response is ready, streaming significantly improves user experience (UX), particularly when dealing with the latency of LLMs.

Overview

LangChain's streaming system lets you surface live feedback from agent runs to your application.

What's possible with LangChain streaming:

- **Stream agent progress** — get state updates after each agent step.
- **Stream LLM tokens** — stream language model tokens as they're generated.
- **Stream custom updates** — emit user-defined signals (e.g., `"Fetched 10/100 records"`).
- **Stream multiple modes** — choose from `updates` (agent progress), `messages` (LLM tokens + metadata), or `custom` (arbitrary user data).

See the **common patterns** section below for additional end-to-end examples.

LangChain Docs

Streaming › **Overview**

Mode	Description
`updates`	Streams state updates after each agent step. If multiple updates are made in the same step (e.g., multiple nodes are run), those updates are streamed separately.
`messages`	Streams tuples of `(token, metadata)` from any graph nodes where an LLM is invoked.
`custom`	Streams custom data from inside your graph nodes using the stream writer.

Agent progress

To stream agent progress, use the `stream` method with `streamMode: "updates"`. This emits an event after every agent step.

For example, if you have an agent that calls a tool once, you should see the following updates:

LLM node: `AIMessage` with tool call requests

Tool node: `ToolMessage` with execution result

LLM node: Final AI response

LangChain Docs

Streaming › **Overview**

LangChain Docs

Streaming › **Overview**

LangChain Docs

Streaming › **Overview**

LangChain Docs

Streaming › **Overview**

LangChain Docs

Streaming › **Overview**

LangChain Docs

Streaming › **Overview**

```
import z from "zod";
import { createAgent, tool } from "langchain";

const getWeather = tool(
    async ({ city }) => {
        return `The weather in ${city} is always sunny!`;
    },
```

LangChain Docs

Streaming › Overview

```
        }),
    }
);

const agent = createAgent({
    model: "gpt-5-nano",
    tools: [getWeather],
});

for await (const chunk of await agent.stream(
    { messages: [{ role: "user", content: "what is the weather in sf" }] },
    { streamMode: "updates" }
)) {
    const [step, content] = Object.entries(chunk)[0];
    console.log(`step: ${step}`);
    console.log(`content: ${JSON.stringify(content, null, 2)}`);
}
/**
 * step: model
 * content: {
 *   "messages": [
 *     {
 *       "kwargs": {
 *         // ...
 *         "tool_calls": [
 *           {
 *             "name": "get_weather",
 *             "args": {
 *               "city": "San Francisco"
 *             },
 *             "type": "tool_call",
 *             "id": "call_0qLS2Jp3MCmaKJ5MAYtr4jJd"
 *           }
 *         ],
 *         // ...
```

```
* step: tools
* content: {
*   "messages": [
*     {
*       "kwargs": {
*         "content": "The weather in San Francisco is always sunny!",
*         "name": "get_weather",
*         // ...
*       }
*     }
*   ]
* }
* step: model
* content: {
*   "messages": [
*     {
*       "kwargs": {
*         "content": "The latest update says: The weather in San Francisco is
*         // ...
*       }
*     }
*   ]
* }
*/
```

LLM tokens

To stream tokens as they are produced by the LLM, use `streamMode: "messages"` :

```
    async ({ city }) => {
        return `The weather in ${city} is always sunny!`;
    },
    {
        name: "get_weather",
        description: "Get weather for a given city.",
        schema: z.object({
        city: z.string(),
        }),
    }
);

const agent = createAgent({
    model: "gpt-4o-mini",
    tools: [getWeather],
});

for await (const [token, metadata] of await agent.stream(
    { messages: [{ role: "user", content: "what is the weather in sf" }] },
    { streamMode: "messages" }
)) {
    console.log(`node: ${metadata.langgraph_node}`);
    console.log(`content: ${JSON.stringify(token.contentBlocks, null, 2)}`);
}
```

Custom updates

To stream updates from tools as they are executed, you can use the `writer` parameter from the configuration.

LangChain Docs

Streaming › Overview

```
const getWeather = tool(
    async (input, config: LangGraphRunnableConfig) => {
        // Stream any arbitrary data
        config.writer?.(`Looking up data for city: ${input.city}`);
        // ... fetch city data
        config.writer?.(`Acquired data for city: ${input.city}`);
        return `It's always sunny in ${input.city}!`;
    },
    {
        name: "get_weather",
        description: "Get weather for a given city.",
        schema: z.object({
        city: z.string().describe("The city to get weather for."),
        }),
    }
);

const agent = createAgent({
    model: "gpt-4o-mini",
    tools: [getWeather],
});

for await (const chunk of await agent.stream(
    { messages: [{ role: "user", content: "what is the weather in sf" }] },
    { streamMode: "custom" }
)) {
    console.log(chunk);
}
```

Output

```
Looking up data for city: San Francisco
quired data for city: San Francisco
```

LangChain Docs

Streaming › **Overview**

Stream multiple modes

You can specify multiple streaming modes by passing streamMode as an array: `streamMode: ["updates", "messages", "custom"]` .

The streamed outputs will be tuples of `[mode, chunk]` where `mode` is the name of the stream mode and `chunk` is the data streamed by that mode.

LangChain Docs

Streaming › Overview

```
const getWeather = tool(
    async (input, config: LangGraphRunnableConfig) => {
        // Stream any arbitrary data
        config.writer?.(`Looking up data for city: ${input.city}`);
        // ... fetch city data
        config.writer?.(`Acquired data for city: ${input.city}`);
        return `It's always sunny in ${input.city}!`;
    },
    {
        name: "get_weather",
        description: "Get weather for a given city.",
        schema: z.object({
        city: z.string().describe("The city to get weather for."),
        }),
    }
);

const agent = createAgent({
    model: "gpt-4o-mini",
    tools: [getWeather],
});

for await (const [streamMode, chunk] of await agent.stream(
    { messages: [{ role: "user", content: "what is the weather in sf" }] },
    { streamMode: ["updates", "messages", "custom"] }
)) {
    console.log(`${streamMode}: ${JSON.stringify(chunk, null, 2)}`);
}
```

Disable streaming

`me applications you might need to disable streaming of individual tokens for a given `el. This is useful when:

streamed to the client

Set `streaming: false` when initializing the model.

```
import { ChatOpenAI } from "@langchain/openai";

const model = new ChatOpenAI({
  model: "gpt-4o",
  streaming: false,
});
```

When deploying to LangSmith, set `streaming=False` on any models whose output you don't want streamed to the client. This is configured in your graph code before deployment.

Not all chat model integrations support the `streaming` parameter. If your model doesn't support it, use `disableStreaming: true` instead. This parameter is available on all chat models via the base class.

See the **LangGraph streaming guide** for more details.

Related

Frontend streaming — Build React UIs with `useStream` for real-time agent interactions

Streaming with chat models — Stream tokens directly from a chat model without using an agent or graph

Streaming with human-in-the-loop — Stream agent progress while handling interrupts for human review

LangGraph streaming — Advanced streaming options including `values`, `debug`

LangChain Docs

Streaming › **Overview**

Edit this page on GitHub or file an issue.

Connect these docs to Claude, VSCode, and more via MCP for real-time answers.

Was this page helpful? Yes No

LangChain Docs

Streaming › **Overview**

Resources

Forum

Changelog

LangChain Academy

Trust Center

Company

About

Careers

Blog

Powered by mintlify

LangChain Docs

Streaming › **Overview**