Streaming

Streaming is an important UX consideration for LLM apps, and agents are no exception. Streaming with agents is made more complicated by the fact that it's not just tokens of the final answer that you will want to stream, but you may also want to stream back the intermediate steps an agent takes.

In this notebook, we'll cover the `stream/astream` and `astream_events` for streaming.

Our agent will use a tools API for tool invocation with the tools:

1. `where_cat_is_hiding`: Returns a location where the cat is hiding
2. `get_items`: Lists items that can be found in a particular place

These tools will allow us to explore streaming in a more interesting situation where the agent will have to use both tools to answer some questions (e.g., to answer the question `what items are located where the cat is hiding?`).

Ready?🏎️

```
from langchain import hub
from langchain.agents import AgentExecutor,
create_openai_tools_agent
from langchain.prompts import
ChatPromptTemplate
from langchain.tools import tool
from langchain_core.callbacks import
Callbacks
from langchain_openai import ChatOpenAI
```

Create the model

Attention We're setting `streaming=True` on the LLM. This will allow us to stream tokens from the agent using the `astream_events` API. This is needed for older versions of LangChain.

```
model = ChatOpenAI(temperature=0,
streaming=True)
```

Tools

We define two tools that rely on a chat model to generate output!

```
import random


@tool
async def where_cat_is_hiding() -> str:
    """Where is the cat hiding right now?"""
    return random.choice(["under the bed",
"on the shelf"])


@tool
async def get_items(place: str) -> str:
    """Use this tool to look up which items
are in the given place."""
    if "bed" in place:  # For under the bed
        return "socks, shoes and dust
bunnies"
    if "shelf" in place:  # For 'shelf'
        return "books, penciles and pictures"
    else:  # if the agent decides to ask
about a different place
        return "cat snacks"
```

```
await where_cat_is_hiding.ainvoke({})
```

```
'on the shelf'
```

```
await get_items.ainvoke({"place": "shelf"})
```

```
'books, penciles and pictures'
```

Initialize the agent

Here, we’ll initialize an OpenAI tools agent.

ATTENTION Please note that we associated the name `Agent` with our agent using `"run_name"="Agent"`. We’ll use that fact later on with the `astream_events` API.

```
# Get the prompt to use - you can modify
this!
prompt = hub.pull("hwchase17/openai-tools-
agent")
# print(prompt.messages) -- to see the prompt
tools = [get_items, where_cat_is_hiding]
agent = create_openai_tools_agent(
    model.with_config({"tags":
["agent_llm"]}), tools, prompt
)
agent_executor = AgentExecutor(agent=agent,
tools=tools).with_config(
    {"run_name": "Agent"}
)
```

Stream Intermediate Steps

We'll use `.stream` method of the AgentExecutor to stream the agent's intermediate steps.

The output from `.stream` alternates between (action, observation) pairs, finally concluding with the answer if the agent achieved its objective.

It'll look like this:

1. actions output
2. observations output
3. actions output
4. observations output

… (continue until goal is reached) …

Then, if the final goal is reached, the agent will output the **final answer**.

The contents of these outputs are summarized here:

Output	Contents
Actions	`actions` `AgentAction` or a subclass, `messages` chat messages corresponding to action invocation
Observations	`steps` History of what the agent did so far, including the current action and its observation, `messages` chat message with function invocation results (aka observations)
Final answer	`output` `AgentFinish`, `messages` chat messages with the final output

```
# Note: We use `pprint` to print only to
depth 1, it makes it easier to see the output
from a high level, before digging in.
import pprint

chunks = []

async for chunk in agent_executor.astream(
    {"input": "what's items are located where
the cat is hiding?"}
):
    chunks.append(chunk)
    print("------")
    pprint.pprint(chunk, depth=1)
```

```
------
{'actions': [...], 'messages': [...]}
------
{'messages': [...], 'steps': [...]}
------
{'actions': [...], 'messages': [...]}
------
{'messages': [...], 'steps': [...]}
------
{'messages': [...],
 'output': 'The items located where the cat
is hiding on the shelf are books, '
           'pencils, and pictures.'}
```

Using Messages

You can access the underlying `messages` from the outputs. Using messages can be nice when working with chat applications - because everything is a message!

```
chunks[0]["actions"]
```

```
[OpenAIToolAgentAction(tool='where_cat_is_hidin
tool_input={}, log='\nInvoking:
`where_cat_is_hiding` with `{}`\n\n\n',
message_log=[AIMessageChunk(content='',
additional_kwargs={'tool_calls': [{'index': 0,
'id': 'call_pKy4OLcBx6pR6k3GHBOlH68r', 'functio
{'arguments': '{}', 'name':
'where_cat_is_hiding'}, 'type': 'function'}]})]
tool_call_id='call_pKy4OLcBx6pR6k3GHBOlH68r')]
```

```
for chunk in chunks:
    print(chunk["messages"])
```

```
[AIMessageChunk(content='',
additional_kwargs={'tool_calls': [{'index':
0, 'id': 'call_pKy4OLcBx6pR6k3GHBOlH68r',
'function': {'arguments': '{}', 'name':
'where_cat_is_hiding'}, 'type':
'function'}]})]
[FunctionMessage(content='on the shelf',
name='where_cat_is_hiding')]
[AIMessageChunk(content='',
additional_kwargs={'tool_calls': [{'index':
0, 'id': 'call_qZTz1mRfCCXT18SUy0E07eS4',
'function': {'arguments': '{\n  "place":
"shelf"\n}', 'name': 'get_items'}, 'type':
'function'}]})]
```

```
[FunctionMessage(content='books, penciles and
pictures', name='get_items')]
[AIMessage(content='The items located where
the cat is hiding on the shelf are books,
pencils, and pictures.')]
```

In addition, they contain full logging information (`actions` and `steps`) which may be easier to process for rendering purposes.

Using AgentAction/Observation

The outputs also contain richer structured information inside of `actions` and `steps`, which could be useful in some situations, but can also be harder to parse.

Attention `AgentFinish` is not available as part of the `streaming` method. If this is something you'd like to be added, please start a discussion on github and explain why its needed.

```
async for chunk in agent_executor.astream(
    {"input": "what's items are located where
the cat is hiding?"}
):
    # Agent Action
    if "actions" in chunk:
        for action in chunk["actions"]:
            print(f"Calling Tool:
`{action.tool}` with input
`{action.tool_input}`")
    # Observation
    elif "steps" in chunk:
        for step in chunk["steps"]:
            print(f"Tool Result:
`{step.observation}`")
    # Final result
    elif "output" in chunk:
        print(f'Final Output:
{chunk["output"]}')
    else:
        raise ValueError()
    print("---")
```

```
Calling Tool: `where_cat_is_hiding` with
input `{}`
---
Tool Result: `on the shelf`
---
Calling Tool: `get_items` with input
`{'place': 'shelf'}`
---
Tool Result: `books, penciles and pictures`
---
Final Output: The items located where the cat
is hiding on the shelf are books, pencils,
and pictures.
---
```

Custom Streaming With Events

Use the `astream_events` API in case the default behavior of *stream* does not work for your application (e.g., if you need to stream individual tokens from the agent or surface steps occuring **within** tools).

⚠️ This is a **beta** API, meaning that some details might change slightly in the future based on usage. ⚠️ To make sure all callbacks work properly, use `async` code throughout. Try avoiding mixing in sync versions of code (e.g., sync versions of tools).

Let’s use this API to stream the following events:

1. Agent Start with inputs
2. Tool Start with inputs
3. Tool End with outputs
4. Stream the agent final anwer token by token
5. Agent End with outputs

```
async for event in
agent_executor.astream_events(
    {"input": "where is the cat hiding? what
items are in that location?"},
    version="v1",
):
    kind = event["event"]
    if kind == "on_chain_start":
        if (
            event["name"] == "Agent"
        ):  # Was assigned when creating the
agent with `.with_config({"run_name":
"Agent"})`
            print(
                f"Starting agent:
{event['name']} with input:
{event['data'].get('input')}"
            )
    elif kind == "on_chain_end":
        if (
            event["name"] == "Agent"
        ):  # Was assigned when creating the
agent with `.with_config({"run_name":
"Agent"})`
            print()
            print("--")
            print(
                f"Done agent: {event['name']}
with output: {event['data'].get('output')
['output']}"
```

```
            )
        if kind == "on_chat_model_stream":
            content = event["data"]
["chunk"].content
            if content:
                # Empty content in the context of
OpenAI means
                # that the model is asking for a
tool to be invoked.
                # So we only print non-empty
content
                print(content, end="|")
        elif kind == "on_tool_start":
            print("--")
            print(
                f"Starting tool: {event['name']}
with inputs: {event['data'].get('input')}"
            )
        elif kind == "on_tool_end":
            print(f"Done tool: {event['name']}")
            print(f"Tool output was:
{event['data'].get('output')}")
            print("--")
```

```
Starting agent: Agent with input: {'input':
'where is the cat hiding? what items are in
that location?'}
--
Starting tool: where_cat_is_hiding with
inputs: {}
```

```
Done tool: where_cat_is_hiding
Tool output was: on the shelf
--
--
Starting tool: get_items with inputs:
{'place': 'shelf'}
Done tool: get_items
Tool output was: books, penciles and pictures
--
The| cat| is| currently| hiding| on| the|
shelf|.| In| that| location|,| you| can|
find| books|,| pencils|,| and| pictures|.|
--
Done agent: Agent with output: The cat is
currently hiding on the shelf. In that
location, you can find books, pencils, and
pictures.
```

Stream Events from within Tools

If your tool leverages LangChain runnable objects (e.g., LCEL chains, LLMs, retrievers etc.) and you want to stream events from those objects as well, you'll need to make sure that callbacks are propagated correctly.

To see how to pass callbacks, let's re-implement the `get_items` tool to make it use an LLM and pass callbacks to that LLM. Feel free to adapt this to your use case.

```
@tool
async def get_items(place: str, callbacks:
Callbacks) -> str:  # <--- Accept callbacks
    """Use this tool to look up which items
are in the given place."""
    template =
ChatPromptTemplate.from_messages(
        [
            (
                "human",
                "Can you tell me what kind of
items i might find in the following place:
'{place}'. "
                "List at least 3 such items
separating them by a comma. And include a
brief description of each item..",
            )
        ]
    )
    chain = template | model.with_config(
        {
            "run_name": "Get Items LLM",
            "tags": ["tool_llm"],
            "callbacks": callbacks,  # <--
Propagate callbacks
        }
    )
    chunks = [chunk async for chunk in
chain.astream({"place": place})]
```

```
    return "".join(chunk.content for chunk in
chunks)
```

^ Take a look at how the tool propagates callbacks.

Next, let’s initialize our agent, and take a look at the new output.

```
# Get the prompt to use - you can modify
this!
prompt = hub.pull("hwchase17/openai-tools-
agent")
# print(prompt.messages) -- to see the prompt
tools = [get_items, where_cat_is_hiding]
agent = create_openai_tools_agent(
    model.with_config({"tags":
["agent_llm"]}), tools, prompt
)
agent_executor = AgentExecutor(agent=agent,
tools=tools).with_config(
    {"run_name": "Agent"}
)

async for event in
agent_executor.astream_events(
    {"input": "where is the cat hiding? what
items are in that location?"},
    version="v1",
):
    kind = event["event"]
    if kind == "on_chain_start":
        if (
            event["name"] == "Agent"
        ):  # Was assigned when creating the
agent with `.with_config({"run_name":
"Agent"})`
            print(
                f"Starting agent:
{event['name']} with input:
{event['data'].get('input')}"
            )
    elif kind == "on_chain_end":
        if (
            event["name"] == "Agent"
        ):  # Was assigned when creating the
agent with `.with_config({"run_name":
"Agent"})`
            print()
            print("--")
            print(
                f"Done agent: {event['name']}
with output: {event['data'].get('output')
['output']}"
            )
    if kind == "on_chat_model_stream":
        content = event["data"]
["chunk"].content
        if content:
            # Empty content in the context of
OpenAI means
```

```
            # that the model is asking for a
tool to be invoked.
            # So we only print non-empty
content
            print(content, end="|")
    elif kind == "on_tool_start":
        print("--")
        print(
            f"Starting tool: {event['name']}
with inputs: {event['data'].get('input')}"
        )
    elif kind == "on_tool_end":
        print(f"Done tool: {event['name']}")
        print(f"Tool output was:
{event['data'].get('output')}")
        print("--")
```

```
Starting agent: Agent with input: {'input':
'where is the cat hiding? what items are in
that location?'}
--
Starting tool: where_cat_is_hiding with
inputs: {}
Done tool: where_cat_is_hiding
Tool output was: on the shelf
--
--
Starting tool: get_items with inputs:
{'place': 'shelf'}
In| a| shelf|,| you| might| find|:
```

```
|1|.| Books|:| A| shelf| is| commonly| used|
to| store| books|.| It| may| contain|
various| genres| such| as| novels|,|
textbooks|,| or| reference| books|.| Books|
provide| knowledge|,| entertainment|,| and|
can| transport| you| to| different| worlds|
through| storytelling|.

|2|.| Decor|ative| items|:| Sh|elves| often|
display| decorative| items| like|
figur|ines|,| v|ases|,| or| photo| frames|.|
These| items| add| a| personal| touch| to|
the| space| and| can| reflect| the| owner|'s|
interests| or| memories|.

|3|.| Storage| boxes|:| Sh|elves| can| also|
hold| storage| boxes| or| baskets|.| These|
containers| help| organize| and| decl|utter|
the| space| by| storing| miscellaneous|
items| like| documents|,| accessories|,| or|
small| household| items|.| They| provide| a|
neat| and| tidy| appearance| to| the|
shelf|.|Done tool: get_items
Tool output was: In a shelf, you might find:

1. Books: A shelf is commonly used to store
books. It may contain various genres such as
novels, textbooks, or reference books. Books
provide knowledge, entertainment, and can
transport you to different worlds through
```

```
storytelling.

2. Decorative items: Shelves often display
decorative items like figurines, vases, or
photo frames. These items add a personal
touch to the space and can reflect the
owner's interests or memories.

3. Storage boxes: Shelves can also hold
storage boxes or baskets. These containers
help organize and declutter the space by
storing miscellaneous items like documents,
accessories, or small household items. They
provide a neat and tidy appearance to the
shelf.
--
The| cat| is| hiding| on| the| shelf|.| In|
that| location|,| you| might| find| books|,|
decorative| items|,| and| storage| boxes|.|
--
Done agent: Agent with output: The cat is
hiding on the shelf. In that location, you
might find books, decorative items, and
storage boxes.
```

Other aproaches

Using astream_log

Note You can also use the astream_log API. This API produces a granular log of all events that occur during execution. The log format is based on the JSONPatch standard. It’s granular, but requires effort to parse. For this reason, we created the `astream_events` API instead.

```
i = 0
async for chunk in
agent_executor.astream_log(
    {"input": "where is the cat hiding? what
items are in that location?"},
):
    print(chunk)
    i += 1
    if i > 10:
        break
```

```
RunLogPatch({'op': 'replace',
  'path': '',
  'value': {'final_output': None,
            'id': 'c261bc30-60d1-4420-9c66-c6c0
            'logs': {},
            'name': 'Agent',
            'streamed_output': [],
            'type': 'chain'}})
RunLogPatch({'op': 'add',
  'path': '/logs/RunnableSequence',
  'value': {'end_time': None,
            'final_output': None,
            'id': '183cb6f8-ed29-4967-b1ea-0240
```

```
                'metadata': {},
                'name': 'RunnableSequence',
                'start_time': '2024-01-22T20:38:43.
                'streamed_output': [],
                'streamed_output_str': [],
                'tags': [],
                'type': 'chain'}})
RunLogPatch({'op': 'add',
  'path': '/logs/RunnableAssign<agent_scratchpa
  'value': {'end_time': None,
                'final_output': None,
                'id': '7fe1bb27-3daf-492e-bc7e-2860
                'metadata': {},
                'name': 'RunnableAssign<agent_scrat
                'start_time': '2024-01-22T20:38:43.
                'streamed_output': [],
                'streamed_output_str': [],
                'tags': ['seq:step:1'],
                'type': 'chain'}})
RunLogPatch({'op': 'add',
  'path':
'/logs/RunnableAssign<agent_scratchpad>/streame
  'value': {'input': 'where is the cat hiding? 
are in that '
                            'location?',
                'intermediate_steps': []}})
RunLogPatch({'op': 'add',
  'path': '/logs/RunnableParallel<agent_scratch
  'value': {'end_time': None,
                'final_output': None,
                'id': 'b034e867-e6bb-4296-bfe6-752c
```

```
            'metadata': {},
            'name': 'RunnableParallel<agent_scr
            'start_time': '2024-01-22T20:38:43.
            'streamed_output': [],
            'streamed_output_str': [],
            'tags': [],
            'type': 'chain'}})
RunLogPatch({'op': 'add',
  'path': '/logs/RunnableLambda',
  'value': {'end_time': None,
            'final_output': None,
            'id': '65ceef3e-7a80-4015-8b5b-d949
            'metadata': {},
            'name': 'RunnableLambda',
            'start_time': '2024-01-22T20:38:43.
            'streamed_output': [],
            'streamed_output_str': [],
            'tags': ['map:key:agent_scratchpad'
            'type': 'chain'}})
RunLogPatch({'op': 'add', 'path':
'/logs/RunnableLambda/streamed_output/-', 'valu
RunLogPatch({'op': 'add',
  'path':
'/logs/RunnableParallel<agent_scratchpad>/strea
',
  'value': {'agent_scratchpad': []}})
RunLogPatch({'op': 'add',
  'path':
'/logs/RunnableAssign<agent_scratchpad>/streame
  'value': {'agent_scratchpad': []}})
RunLogPatch({'op': 'add',
  'path': '/logs/RunnableLambda/final_output',
  'value': {'output': []}},
 {'op': 'add',
  'path': '/logs/RunnableLambda/end_time',
  'value': '2024-01-22T20:38:43.654+00:00'})
RunLogPatch({'op': 'add',
  'path':
'/logs/RunnableParallel<agent_scratchpad>/final_
  'value': {'agent_scratchpad': []}},
 {'op': 'add',
  'path':
'/logs/RunnableParallel<agent_scratchpad>/end_t
  'value': '2024-01-22T20:38:43.655+00:00'})
```

This may require some logic to get in a workable format

```
i = 0
path_status = {}
async for chunk in
agent_executor.astream_log(
    {"input": "where is the cat hiding? what
items are in that location?"},
):
    for op in chunk.ops:
        if op["op"] == "add":
            if op["path"] not in path_status:
                path_status[op["path"]] =
op["value"]
            else:
                path_status[op["path"]] +=
op["value"]
    print(op["path"])
    print(path_status.get(op["path"]))
    print("----")
    i += 1
    if i > 30:
        break
```

```
None
----
/logs/RunnableSequence
{'id': '22bbd5db-9578-4e3f-a6ec-9b61f08cb8a9',
'RunnableSequence', 'type': 'chain', 'tags': []
'metadata': {}, 'start_time': '2024-01-
22T20:38:43.668+00:00', 'streamed_output': [],
'streamed_output_str': [], 'final_output': None
'end_time': None}
----
/logs/RunnableAssign<agent_scratchpad>
{'id': 'e0c00ae2-aaa2-4a09-bc93-cb34bf3f6554',
'RunnableAssign<agent_scratchpad>', 'type': 'ch
'tags': ['seq:step:1'], 'metadata': {}, 'start_
'2024-01-22T20:38:43.672+00:00', 'streamed_outp
'streamed_output_str': [], 'final_output': None
'end_time': None}
----
/logs/RunnableAssign<agent_scratchpad>/streamed
{'input': 'where is the cat hiding? what items
```

```
location?', 'intermediate_steps': []}
----
/logs/RunnableParallel<agent_scratchpad>
{'id': '26ff576d-ff9d-4dea-98b2-943312a37f4d',
'RunnableParallel<agent_scratchpad>', 'type': '
'tags': [], 'metadata': {}, 'start_time': '2024
22T20:38:43.674+00:00', 'streamed_output': [],
'streamed_output_str': [], 'final_output': None
'end_time': None}
----
/logs/RunnableLambda
{'id': '9f343c6a-23f7-4a28-832f-d4fe3e95d1dc',
'RunnableLambda', 'type': 'chain', 'tags':
['map:key:agent_scratchpad'], 'metadata': {},
'start_time': '2024-01-22T20:38:43.685+00:00',
'streamed_output': [], 'streamed_output_str': [
'final_output': None, 'end_time': None}
----
/logs/RunnableLambda/streamed_output/-
[]
----
/logs/RunnableParallel<agent_scratchpad>/stream
{'agent_scratchpad': []}
----
/logs/RunnableAssign<agent_scratchpad>/streamed
{'input': 'where is the cat hiding? what items
location?', 'intermediate_steps': [], 'agent_sc
[]}
----
/logs/RunnableLambda/end_time
2024-01-22T20:38:43.687+00:00
```

```
----
/logs/RunnableParallel<agent_scratchpad>/end_ti
2024-01-22T20:38:43.688+00:00
----
/logs/RunnableAssign<agent_scratchpad>/end_time
2024-01-22T20:38:43.688+00:00
----
/logs/ChatPromptTemplate
{'id': '7e3a84d5-46b8-4782-8eed-d1fe92be6a30',
'ChatPromptTemplate', 'type': 'prompt', 'tags':
['seq:step:2'], 'metadata': {}, 'start_time': '
22T20:38:43.689+00:00', 'streamed_output': [],
'streamed_output_str': [], 'final_output': None
'end_time': None}
----
/logs/ChatPromptTemplate/end_time
2024-01-22T20:38:43.689+00:00
----
/logs/ChatOpenAI
{'id': '6446f7ec-b3e4-4637-89d8-b4b34b46ea14',
'ChatOpenAI', 'type': 'llm', 'tags': ['seq:step
'agent_llm'], 'metadata': {}, 'start_time': '20
22T20:38:43.690+00:00', 'streamed_output': [],
'streamed_output_str': [], 'final_output': None
'end_time': None}
----
/logs/ChatOpenAI/streamed_output/-
content='' additional_kwargs={'tool_calls': [{'
'id': 'call_gKFg6FX8ZQ88wFUs94yx86PF', 'functio
{'arguments': '', 'name': 'where_cat_is_hiding'
'function'}]}
```

```
----
/logs/ChatOpenAI/streamed_output/-
content='' additional_kwargs={'tool_calls': [{'
'id': 'call_gKFg6FX8ZQ88wFUs94yx86PF', 'functio
{'arguments': '{}', 'name': 'where_cat_is_hidin
'type': 'function'}]}
----
/logs/ChatOpenAI/streamed_output/-
content='' additional_kwargs={'tool_calls': [{'
'id': 'call_gKFg6FX8ZQ88wFUs94yx86PF', 'functio
{'arguments': '{}', 'name': 'where_cat_is_hidin
'type': 'function'}]}
----
/logs/ChatOpenAI/end_time
2024-01-22T20:38:44.203+00:00
----
/logs/OpenAIToolsAgentOutputParser
{'id': '65912835-8dcd-4be2-ad05-9f239a7ef704',
'OpenAIToolsAgentOutputParser', 'type': 'parser
['seq:step:4'], 'metadata': {}, 'start_time': '
22T20:38:44.204+00:00', 'streamed_output': [],
'streamed_output_str': [], 'final_output': None
'end_time': None}
----
/logs/OpenAIToolsAgentOutputParser/end_time
2024-01-22T20:38:44.205+00:00
----
/logs/RunnableSequence/streamed_output/-
[OpenAIToolAgentAction(tool='where_cat_is_hidin
tool_input={}, log='\nInvoking: `where_cat_is_h
`{}`\n\n\n', message_log=[AIMessageChunk(conten
additional_kwargs={'tool_calls': [{'index': 0,
'call_gKFg6FX8ZQ88wFUs94yx86PF', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type':
'function'}]})],
tool_call_id='call_gKFg6FX8ZQ88wFUs94yx86PF')]
----
/logs/RunnableSequence/end_time
2024-01-22T20:38:44.206+00:00
----
/final_output
None
----
/logs/where_cat_is_hiding
{'id': '21fde139-0dfa-42bb-ad90-b5b1e984aaba',
'where_cat_is_hiding', 'type': 'tool', 'tags':
'metadata': {}, 'start_time': '2024-01-
22T20:38:44.208+00:00', 'streamed_output': [],
'streamed_output_str': [], 'final_output': None
'end_time': None}
----
/logs/where_cat_is_hiding/end_time
2024-01-22T20:38:44.208+00:00
----
/final_output/messages/1
content='under the bed' name='where_cat_is_hidi
----
/logs/RunnableSequence:2
{'id': '37d52845-b689-4c18-9c10-ffdd0c4054b0',
'RunnableSequence', 'type': 'chain', 'tags': []
'metadata': {}, 'start_time': '2024-01-
22T20:38:44.210+00:00', 'streamed_output': [],
'streamed_output_str': [], 'final_output': None
'end_time': None}
----
/logs/RunnableAssign<agent_scratchpad>:2
{'id': '30024dea-064f-4b04-b130-671f47ac59bc',
'RunnableAssign<agent_scratchpad>', 'type': 'ch
'tags': ['seq:step:1'], 'metadata': {}, 'start_
'2024-01-22T20:38:44.213+00:00', 'streamed_outp
'streamed_output_str': [], 'final_output': None
'end_time': None}
----
/logs/RunnableAssign<agent_scratchpad>:2/stream
{'input': 'where is the cat hiding? what items
location?', 'intermediate_steps':
[(OpenAIToolAgentAction(tool='where_cat_is_hidi
tool_input={}, log='\nInvoking: `where_cat_is_h
`{}`\n\n\n', message_log=[AIMessageChunk(conten
additional_kwargs={'tool_calls': [{'index': 0,
'call_gKFg6FX8ZQ88wFUs94yx86PF', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type':
'function'}]})],
tool_call_id='call_gKFg6FX8ZQ88wFUs94yx86PF'),
bed')]}
----
/logs/RunnableParallel<agent_scratchpad>:2
{'id': '98906cd7-93c2-47e8-a7d7-2e8d4ab09ed0',
'RunnableParallel<agent_scratchpad>', 'type': '
'tags': [], 'metadata': {}, 'start_time': '2024
22T20:38:44.215+00:00', 'streamed_output': [],
'streamed_output_str': [], 'final_output': None
```

```
'end_time': None}
----
```

Using callbacks (Legacy)

Another approach to streaming is using callbacks. This may be useful if you’re still on an older version of LangChain and cannot upgrade.

Generall, this is **NOT** a recommended approach because:

1. for most applications, you’ll need to create two workers, write the callbacks to a queue and have another worker reading from the queue (i.e., there’s hidden complexity to make this work).
2. **end** events may be missing some metadata (e.g., like run name). So if you need the additional metadata, you should inherit from `BaseTracer` instead of `AsyncCallbackHandler` to pick up the relevant information from the runs (aka traces), or else implement the aggregation logic yourself based on the `run_id`.
3. There is inconsistent behavior with the callbacks (e.g., how inputs and outputs are encoded) depending on the callback type that you’ll need to workaround.

For illustration purposes, we implement a callback below that shows how to get *token by token* streaming. Feel free to implement other callbacks based on your application needs.

But `astream_events` does all of this you under the hood, so you don't have to!

```
from typing import TYPE_CHECKING, Any, Dict,
List, Optional, Sequence, TypeVar, Union
from uuid import UUID

from langchain_core.callbacks.base import
AsyncCallbackHandler
from langchain_core.messages import
BaseMessage
from langchain_core.outputs import
ChatGenerationChunk, GenerationChunk,
LLMResult

# Here is a custom handler that will print
the tokens to stdout.
# Instead of printing to stdout you can send
the data elsewhere; e.g., to a streaming API
response


class
TokenByTokenHandler(AsyncCallbackHandler):
    def __init__(self, tags_of_interest:
List[str]) -> None:
        """A custom call back handler.
        Args:
            tags_of_interest: Only LLM tokens
from models with these tags will be
                              printed.
        """
        self.tags_of_interest =
tags_of_interest

    async def on_chain_start(
        self,
        serialized: Dict[str, Any],
        inputs: Dict[str, Any],
        *,
        run_id: UUID,
        parent_run_id: Optional[UUID] = None,
        tags: Optional[List[str]] = None,
        metadata: Optional[Dict[str, Any]] =
None,
        **kwargs: Any,
    ) -> None:
        """Run when chain starts running."""
        print("on chain start: ")
        print(inputs)

    async def on_chain_end(
        self,
        outputs: Dict[str, Any],
        *,
        run_id: UUID,
        parent_run_id: Optional[UUID] = None,
        tags: Optional[List[str]] = None,
        **kwargs: Any,
    ) -> None:
        """Run when chain ends running."""
        print("On chain end")
        print(outputs)

    async def on_chat_model_start(
        self,
        serialized: Dict[str, Any],
        messages: List[List[BaseMessage]],
        *,
        run_id: UUID,
        parent_run_id: Optional[UUID] = None,
        tags: Optional[List[str]] = None,
        metadata: Optional[Dict[str, Any]] =
None,
        **kwargs: Any,
    ) -> Any:
        """Run when a chat model starts
running."""
        overlap_tags =
self.get_overlap_tags(tags)

        if overlap_tags:
            print(",".join(overlap_tags),
end=": ", flush=True)

    def on_tool_start(
        self,
        serialized: Dict[str, Any],
        input_str: str,
        *,
        run_id: UUID,
        parent_run_id: Optional[UUID] = None,
        tags: Optional[List[str]] = None,
        metadata: Optional[Dict[str, Any]] =
None,
        inputs: Optional[Dict[str, Any]] =
None,
        **kwargs: Any,
    ) -> Any:
        """Run when tool starts running."""
        print("Tool start")
        print(serialized)

    def on_tool_end(
        self,
        output: str,
        *,
        run_id: UUID,
        parent_run_id: Optional[UUID] = None,
        **kwargs: Any,
    ) -> Any:
        """Run when tool ends running."""
        print("Tool end")
        print(output)

    async def on_llm_end(
        self,
        response: LLMResult,
        *,
        run_id: UUID,
        parent_run_id: Optional[UUID] = None,
        tags: Optional[List[str]] = None,
        **kwargs: Any,
    ) -> None:
        """Run when LLM ends running."""
        overlap_tags =
self.get_overlap_tags(tags)

        if overlap_tags:
            # Who can argue with beauty?
            print()
            print()

    def get_overlap_tags(self, tags:
Optional[List[str]]) -> List[str]:
        """Check for overlap with filtered
tags."""
        if not tags:
            return []
        return sorted(set(tags or []) &
set(self.tags_of_interest or []))

    async def on_llm_new_token(
        self,
        token: str,
        *,
        chunk:
Optional[Union[GenerationChunk,
ChatGenerationChunk]] = None,
        run_id: UUID,
        parent_run_id: Optional[UUID] = None,
        tags: Optional[List[str]] = None,
        **kwargs: Any,
    ) -> None:
        """Run on new LLM token. Only
available when streaming is enabled."""
        overlap_tags =
self.get_overlap_tags(tags)

        if token and overlap_tags:
            print(token, end="|", flush=True)


handler =
TokenByTokenHandler(tags_of_interest=
["tool_llm", "agent_llm"])

result = await agent_executor.ainvoke(
    {"input": "where is the cat hiding and
what items can be found there?"},
    {"callbacks": [handler]},
)
```

```
on chain start:
{'input': 'where is the cat hiding and what ite
there?'}
on chain start:
{'input': ''}
on chain start:
{'input': ''}
on chain start:
```

```
{'input': ''}
on chain start:
{'input': ''}
On chain end
[]
On chain end
{'agent_scratchpad': []}
On chain end
{'input': 'where is the cat hiding and what ite
there?', 'intermediate_steps': [], 'agent_scrat
on chain start:
{'input': 'where is the cat hiding and what ite
there?', 'intermediate_steps': [], 'agent_scrat
On chain end
{'lc': 1, 'type': 'constructor', 'id': ['langch
'prompts', 'chat', 'ChatPromptValue'], 'kwargs'
[{'lc': 1, 'type': 'constructor', 'id': ['langc
'schema', 'messages', 'SystemMessage'], 'kwargs
'You are a helpful assistant', 'additional_kwar
{'lc': 1, 'type': 'constructor', 'id': ['langch
'messages', 'HumanMessage'], 'kwargs': {'conten
the cat hiding and what items can be found ther
'additional_kwargs': {}}}]}}
agent_llm:

on chain start:
content='' additional_kwargs={'tool_calls': [{'
'id': 'call_pboyZTT0587rJtujUluO2OOc', 'functio
{'arguments': '{}', 'name': 'where_cat_is_hidin
'function'}]}
On chain end
```

```
[{'lc': 1, 'type': 'constructor', 'id': ['langc
'schema', 'agent', 'OpenAIToolAgentAction'], 'k
{'tool': 'where_cat_is_hiding', 'tool_input': {
'\nInvoking: `where_cat_is_hiding` with `{}`\n\
'message_log': [{'lc': 1, 'type': 'constructor'
['langchain', 'schema', 'messages', 'AIMessageC
'kwargs': {'example': False, 'content': '',
'additional_kwargs': {'tool_calls': [{'index':
'call_pboyZTT0587rJtujUlu02OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type':
'function'}]}}}], 'tool_call_id':
'call_pboyZTT0587rJtujUlu02OOc'}}]
On chain end
[OpenAIToolAgentAction(tool='where_cat_is_hidin
{}, log='\nInvoking: `where_cat_is_hiding` with
message_log=[AIMessageChunk(content='', additio
{'tool_calls': [{'index': 0, 'id':
'call_pboyZTT0587rJtujUlu02OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
tool_call_id='call_pboyZTT0587rJtujUlu02OOc')]
Tool start
{'name': 'where_cat_is_hiding', 'description':
'where_cat_is_hiding() -> str - Where is the ca
now?'}
Tool end
on the shelf
on chain start:
{'input': ''}
on chain start:
{'input': ''}
on chain start:
```

```
{'input': ''}
on chain start:
{'input': ''}
On chain end
[AIMessageChunk(content='', additional_kwargs={
[{'index': 0, 'id': 'call_pboyZTT0587rJtujUluO2
'function': {'arguments': '{}', 'name':
'where_cat_is_hiding'}, 'type': 'function'}]}),
ToolMessage(content='on the shelf', additional_
'where_cat_is_hiding'},
tool_call_id='call_pboyZTT0587rJtujUluO2OOc')]
On chain end
{'agent_scratchpad': [AIMessageChunk(content=''
additional_kwargs={'tool_calls': [{'index': 0,
'call_pboyZTT0587rJtujUluO2OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
ToolMessage(content='on the shelf', additional_
'where_cat_is_hiding'},
tool_call_id='call_pboyZTT0587rJtujUluO2OOc')]}
On chain end
{'input': 'where is the cat hiding and what ite
there?', 'intermediate_steps':
[(OpenAIToolAgentAction(tool='where_cat_is_hidi
tool_input={}, log='\nInvoking: `where_cat_is_h
`{}`\n\n\n', message_log=[AIMessageChunk(conten
additional_kwargs={'tool_calls': [{'index': 0,
'call_pboyZTT0587rJtujUluO2OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
tool_call_id='call_pboyZTT0587rJtujUluO2OOc'),
shelf')], 'agent_scratchpad': [AIMessageChunk(c
additional_kwargs={'tool_calls': [{'index': 0,
'call_pboyZTT0587rJtujUluO2OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
ToolMessage(content='on the shelf', additional_
'where_cat_is_hiding'},
tool_call_id='call_pboyZTT0587rJtujUluO2OOc')]}
on chain start:
{'input': 'where is the cat hiding and what ite
there?', 'intermediate_steps':
[(OpenAIToolAgentAction(tool='where_cat_is_hidi
tool_input={}, log='\nInvoking: `where_cat_is_h
`{}`\n\n\n', message_log=[AIMessageChunk(conten
additional_kwargs={'tool_calls': [{'index': 0,
'call_pboyZTT0587rJtujUluO2OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
tool_call_id='call_pboyZTT0587rJtujUluO2OOc'),
shelf')], 'agent_scratchpad': [AIMessageChunk(c
additional_kwargs={'tool_calls': [{'index': 0,
'call_pboyZTT0587rJtujUluO2OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
ToolMessage(content='on the shelf', additional_
'where_cat_is_hiding'},
tool_call_id='call_pboyZTT0587rJtujUluO2OOc')]}
On chain end
{'lc': 1, 'type': 'constructor', 'id': ['langch
'prompts', 'chat', 'ChatPromptValue'], 'kwargs'
[{'lc': 1, 'type': 'constructor', 'id': ['langc
'schema', 'messages', 'SystemMessage'], 'kwargs
'You are a helpful assistant', 'additional_kwar
{'lc': 1, 'type': 'constructor', 'id': ['langch
'messages', 'HumanMessage'], 'kwargs': {'conten
the cat hiding and what items can be found ther
```

```
'additional_kwargs': {}}}, {'lc': 1, 'type': 'c
'id': ['langchain', 'schema', 'messages', 'AIMe
'kwargs': {'example': False, 'content': '',
'additional_kwargs': {'tool_calls': [{'index': 
'call_pboyZTT0587rJtujUluO2OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
{'lc': 1, 'type': 'constructor', 'id': ['langch
'messages', 'ToolMessage'], 'kwargs': {'tool_ca
'call_pboyZTT0587rJtujUluO2OOc', 'content': 'on
'additional_kwargs': {'name': 'where_cat_is_hid
agent_llm:

on chain start:
content='' additional_kwargs={'tool_calls': [{'
'id': 'call_vIVtgUb9Gvmc3zAGIrshnmbh', 'functio
{'arguments': '{\n  "place": "shelf"\n}', 'name
'get_items'}, 'type': 'function'}]}
On chain end
[{'lc': 1, 'type': 'constructor', 'id': ['langc
'schema', 'agent', 'OpenAIToolAgentAction'], 'k
{'tool': 'get_items', 'tool_input': {'place': '
"\nInvoking: `get_items` with `{'place': 'shelf
'message_log': [{'lc': 1, 'type': 'constructor'
['langchain', 'schema', 'messages', 'AIMessageC
'kwargs': {'example': False, 'content': '',
'additional_kwargs': {'tool_calls': [{'index': 
'call_vIVtgUb9Gvmc3zAGIrshnmbh', 'function': {'
'{\n  "place": "shelf"\n}', 'name': 'get_items'
'function'}]}}}], 'tool_call_id':
'call_vIVtgUb9Gvmc3zAGIrshnmbh'}}]
On chain end
```

```
[OpenAIToolAgentAction(tool='get_items', tool_i
'shelf'}, log="\nInvoking: `get_items` with `{'
'shelf'}`\n\n\n", message_log=[AIMessageChunk(c
additional_kwargs={'tool_calls': [{'index': 0,
'call_vIVtgUb9Gvmc3zAGIrshnmbh', 'function': {'
'{\n  "place": "shelf"\n}', 'name': 'get_items'
'function'}]})],
tool_call_id='call_vIVtgUb9Gvmc3zAGIrshnmbh')]
Tool start
{'name': 'get_items', 'description': 'get_items
callbacks:
Union[List[langchain_core.callbacks.base.BaseCa
langchain_core.callbacks.base.BaseCallbackManag
-> str - Use this tool to look up which items a
place.'}
tool_llm: In| a| shelf|,| you| might| find|:

|1|.| Books|:| A| shelf| is| commonly| used| to
books|.| Books| can| be| of| various| genres|,|
novels|,| textbooks|,| or| reference| books|.|
knowledge|,| entertainment|,| and| can| transpo
different| worlds| through| storytelling|.

|2|.| Decor|ative| items|:| Sh|elves| often| se
display| area| for| decorative| items| like| fi
v|ases|,| or| sculptures|.| These| items| add|
value| to| the| space| and| reflect| the| owner
taste| and| style|.

|3|.| Storage| boxes|:| Sh|elves| can| also| be
store| various| items| in| organized| boxes|.|
can| hold| anything| from| office| supplies|,| c
materials|,| or| sentimental| items|.| They| he
space| tidy| and| provide| easy| access| to| st
belongings|.|

Tool end
In a shelf, you might find:

1. Books: A shelf is commonly used to store boo
be of various genres, such as novels, textbooks
books. They provide knowledge, entertainment, a
transport you to different worlds through story

2. Decorative items: Shelves often serve as a d
decorative items like figurines, vases, or scul
items add aesthetic value to the space and refl
personal taste and style.

3. Storage boxes: Shelves can also be used to s
items in organized boxes. These boxes can hold 
office supplies, craft materials, or sentimenta
help keep the space tidy and provide easy acces
belongings.
on chain start:
{'input': ''}
on chain start:
{'input': ''}
on chain start:
{'input': ''}
on chain start:
{'input': ''}
```

```
On chain end
[AIMessageChunk(content='', additional_kwargs={
[{'index': 0, 'id': 'call_pboyZTT0587rJtujUluO2
'function': {'arguments': '{}', 'name':
'where_cat_is_hiding'}, 'type': 'function'}]}),
ToolMessage(content='on the shelf', additional_
'where_cat_is_hiding'},
tool_call_id='call_pboyZTT0587rJtujUluO2OOc'),
AIMessageChunk(content='', additional_kwargs={'
[{'index': 0, 'id': 'call_vIVtgUb9Gvmc3zAGIrshn
'function': {'arguments': '{\n  "place": "shelf
'get_items'}, 'type': 'function'}]}), ToolMessa
a shelf, you might find:\n\n1. Books: A shelf i
to store books. Books can be of various genres,
novels, textbooks, or reference books. They pro
entertainment, and can transport you to differe
through storytelling.\n\n2. Decorative items: S
serve as a display area for decorative items li
vases, or sculptures. These items add aesthetic
space and reflect the owner's personal taste an
Storage boxes: Shelves can also be used to stor
in organized boxes. These boxes can hold anythi
supplies, craft materials, or sentimental items
keep the space tidy and provide easy access to
belongings.", additional_kwargs={'name': 'get_i
tool_call_id='call_vIVtgUb9Gvmc3zAGIrshnmbh')]
On chain end
{'agent_scratchpad': [AIMessageChunk(content=''
additional_kwargs={'tool_calls': [{'index': 0,
'call_pboyZTT0587rJtujUluO2OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
```

```
ToolMessage(content='on the shelf', additional_
'where_cat_is_hiding'},
tool_call_id='call_pboyZTT0587rJtujUlu02OOc'),
AIMessageChunk(content='', additional_kwargs={'
[{'index': 0, 'id': 'call_vIVtgUb9Gvmc3zAGIrshn
'function': {'arguments': '{\n  "place": "shelf
'get_items'}, 'type': 'function'}]}), ToolMessa
a shelf, you might find:\n\n1. Books: A shelf i
to store books. Books can be of various genres,
novels, textbooks, or reference books. They pro
entertainment, and can transport you to differe
through storytelling.\n\n2. Decorative items: S
serve as a display area for decorative items li
vases, or sculptures. These items add aesthetic
space and reflect the owner's personal taste an
Storage boxes: Shelves can also be used to stor
in organized boxes. These boxes can hold anythi
supplies, craft materials, or sentimental items
keep the space tidy and provide easy access to 
belongings.", additional_kwargs={'name': 'get_i
tool_call_id='call_vIVtgUb9Gvmc3zAGIrshnmbh')]}
On chain end
{'input': 'where is the cat hiding and what ite
there?', 'intermediate_steps':
[(OpenAIToolAgentAction(tool='where_cat_is_hidi
tool_input={}, log='\nInvoking: `where_cat_is_h
`{}`\n\n\n', message_log=[AIMessageChunk(conten
additional_kwargs={'tool_calls': [{'index': 0, 
'call_pboyZTT0587rJtujUlu02OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
tool_call_id='call_pboyZTT0587rJtujUlu02OOc'),
shelf'), (OpenAIToolAgentAction(tool='get_items
{'place': 'shelf'}, log="\nInvoking: `get_items
`{'place': 'shelf'}`\n\n\n", message_log=
[AIMessageChunk(content='', additional_kwargs={
[{'index': 0, 'id': 'call_vIVtgUb9Gvmc3zAGIrshn
'function': {'arguments': '{\n  "place": "shelf
'get_items'}, 'type': 'function'}]})],
tool_call_id='call_vIVtgUb9Gvmc3zAGIrshnmbh'),
you might find:\n\n1. Books: A shelf is commonl
books. Books can be of various genres, such as 
textbooks, or reference books. They provide kno
entertainment, and can transport you to differe
through storytelling.\n\n2. Decorative items: S
serve as a display area for decorative items li
vases, or sculptures. These items add aesthetic
space and reflect the owner's personal taste an
Storage boxes: Shelves can also be used to stor
in organized boxes. These boxes can hold anythi
supplies, craft materials, or sentimental items
keep the space tidy and provide easy access to 
belongings.")], 'agent_scratchpad':
[AIMessageChunk(content='', additional_kwargs={
[{'index': 0, 'id': 'call_pboyZTT0587rJtujUluO2
'function': {'arguments': '{}', 'name':
'where_cat_is_hiding'}, 'type': 'function'}]}),
ToolMessage(content='on the shelf', additional_
'where_cat_is_hiding'},
tool_call_id='call_pboyZTT0587rJtujUluO2OOc'),
AIMessageChunk(content='', additional_kwargs={'
[{'index': 0, 'id': 'call_vIVtgUb9Gvmc3zAGIrshn
'function': {'arguments': '{\n  "place": "shelf
```

```
'get_items'}, 'type': 'function'}]}), ToolMessa
a shelf, you might find:\n\n1. Books: A shelf i
to store books. Books can be of various genres,
novels, textbooks, or reference books. They pro
entertainment, and can transport you to differe
through storytelling.\n\n2. Decorative items: S
serve as a display area for decorative items li
vases, or sculptures. These items add aesthetic
space and reflect the owner's personal taste an
Storage boxes: Shelves can also be used to stor
in organized boxes. These boxes can hold anythi
supplies, craft materials, or sentimental items
keep the space tidy and provide easy access to
belongings.", additional_kwargs={'name': 'get_i
tool_call_id='call_vIVtgUb9Gvmc3zAGIrshnmbh')]}
on chain start:
{'input': 'where is the cat hiding and what ite
there?', 'intermediate_steps':
[(OpenAIToolAgentAction(tool='where_cat_is_hidi
tool_input={}, log='\nInvoking: `where_cat_is_h
`{}`\n\n\n', message_log=[AIMessageChunk(conten
additional_kwargs={'tool_calls': [{'index': 0,
'call_pboyZTT0587rJtujUluO2OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
tool_call_id='call_pboyZTT0587rJtujUluO2OOc'),
shelf'), (OpenAIToolAgentAction(tool='get_items
{'place': 'shelf'}, log="\nInvoking: `get_items
`{'place': 'shelf'}`\n\n\n", message_log=
[AIMessageChunk(content='', additional_kwargs={
[{'index': 0, 'id': 'call_vIVtgUb9Gvmc3zAGIrshn
'function': {'arguments': '{\n  "place": "shelf
'get_items'}, 'type': 'function'}]})],
tool_call_id='call_vIVtgUb9Gvmc3zAGIrshnmbh'),
you might find:\n\n1. Books: A shelf is commonl
books. Books can be of various genres, such as
textbooks, or reference books. They provide kno
entertainment, and can transport you to differe
through storytelling.\n\n2. Decorative items: S
serve as a display area for decorative items li
vases, or sculptures. These items add aesthetic
space and reflect the owner's personal taste an
Storage boxes: Shelves can also be used to stor
in organized boxes. These boxes can hold anythi
supplies, craft materials, or sentimental items
keep the space tidy and provide easy access to
belongings.")], 'agent_scratchpad':
[AIMessageChunk(content='', additional_kwargs={
[{'index': 0, 'id': 'call_pboyZTT0587rJtujUluO2
'function': {'arguments': '{}', 'name':
'where_cat_is_hiding'}, 'type': 'function'}]}),
ToolMessage(content='on the shelf', additional_
'where_cat_is_hiding'},
tool_call_id='call_pboyZTT0587rJtujUluO2OOc'),
AIMessageChunk(content='', additional_kwargs={'
[{'index': 0, 'id': 'call_vIVtgUb9Gvmc3zAGIrshn
'function': {'arguments': '{\n  "place": "shelf
'get_items'}, 'type': 'function'}]}), ToolMessa
a shelf, you might find:\n\n1. Books: A shelf i
to store books. Books can be of various genres,
novels, textbooks, or reference books. They pro
entertainment, and can transport you to differe
through storytelling.\n\n2. Decorative items: S
```

```
serve as a display area for decorative items li
vases, or sculptures. These items add aesthetic
space and reflect the owner's personal taste an
Storage boxes: Shelves can also be used to stor
in organized boxes. These boxes can hold anythi
supplies, craft materials, or sentimental items
keep the space tidy and provide easy access to 
belongings.", additional_kwargs={'name': 'get_i
tool_call_id='call_vIVtgUb9Gvmc3zAGIrshnmbh')]}
On chain end
{'lc': 1, 'type': 'constructor', 'id': ['langch
'prompts', 'chat', 'ChatPromptValue'], 'kwargs'
[{'lc': 1, 'type': 'constructor', 'id': ['langc
'schema', 'messages', 'SystemMessage'], 'kwargs
'You are a helpful assistant', 'additional_kwar
{'lc': 1, 'type': 'constructor', 'id': ['langch
'messages', 'HumanMessage'], 'kwargs': {'conten
the cat hiding and what items can be found ther
'additional_kwargs': {}}}, {'lc': 1, 'type': 'c
'id': ['langchain', 'schema', 'messages', 'AIMe
'kwargs': {'example': False, 'content': '',
'additional_kwargs': {'tool_calls': [{'index': 
'call_pboyZTT0587rJtujUlu02OOc', 'function': {'
'{}', 'name': 'where_cat_is_hiding'}, 'type': '
{'lc': 1, 'type': 'constructor', 'id': ['langch
'messages', 'ToolMessage'], 'kwargs': {'tool_ca
'call_pboyZTT0587rJtujUlu02OOc', 'content': 'on
'additional_kwargs': {'name': 'where_cat_is_hid
1, 'type': 'constructor', 'id': ['langchain', '
'messages', 'AIMessageChunk'], 'kwargs': {'exam
'content': '', 'additional_kwargs': {'tool_call
```

```
0, 'id': 'call_vIVtgUb9Gvmc3zAGIrshnmbh', 'func
{'arguments': '{\n  "place": "shelf"\n}', 'name
'get_items'}, 'type': 'function'}]}}}, {'lc': 1
'constructor', 'id': ['langchain', 'schema', 'm
'ToolMessage'], 'kwargs': {'tool_call_id':
'call_vIVtgUb9Gvmc3zAGIrshnmbh', 'content': "In
might find:\n\n1. Books: A shelf is commonly us
books. Books can be of various genres, such as
textbooks, or reference books. They provide kno
entertainment, and can transport you to differe
through storytelling.\n\n2. Decorative items: S
serve as a display area for decorative items li
vases, or sculptures. These items add aesthetic
space and reflect the owner's personal taste an
Storage boxes: Shelves can also be used to stor
in organized boxes. These boxes can hold anythi
supplies, craft materials, or sentimental items
keep the space tidy and provide easy access to
belongings.", 'additional_kwargs': {'name': 'ge
agent_llm: The| cat| is| hiding| on| the| shelf
shelf|,| you| might| find| books|,| decorative|
storage| boxes|.|

on chain start:
content='The cat is hiding on the shelf. In the
might find books, decorative items, and storage
On chain end
{'lc': 1, 'type': 'constructor', 'id': ['langch
'agent', 'AgentFinish'], 'kwargs': {'return_val
{'output': 'The cat is hiding on the shelf. In
might find books, decorative items, and storage
```

```
'log': 'The cat is hiding on the shelf. In the
might find books, decorative items, and storage
On chain end
return_values={'output': 'The cat is hiding on
the shelf, you might find books, decorative iter
boxes.'} log='The cat is hiding on the shelf. I
you might find books, decorative items, and sto
On chain end
{'output': 'The cat is hiding on the shelf. In
might find books, decorative items, and storage
```