

[Modules](#)[Agents](#)[Quickstart](#)

# Quickstart

To best understand the agent framework, let's build an agent that has two tools: one to look things up online, and one to look up specific data that we've loaded into a index.

This will assume knowledge of [LLMs](#) and [retrieval](#) so if you haven't already explored those sections, it is recommended you do so.

## Setup: LangSmith

By definition, agents take a self-determined, input-dependent sequence of steps before returning a user-facing output. This makes debugging these systems particularly tricky, and observability particularly important. [LangSmith](#) is especially useful for such cases.

When building with LangChain, all steps will automatically be traced in LangSmith. To set up LangSmith we just need set the following environment variables:

```
export LANGCHAIN_TRACING_V2="true"  
export LANGCHAIN_API_KEY="<your-api-key>"
```

## Define tools

We first need to create the tools we want to use. We will use two tools: **Tavily** (to search online) and then a retriever over a local index we will create

### Tavily

We have a built-in tool in LangChain to easily use Tavily search engine as tool. Note that this requires an API key - they have a free tier, but if you don't have one or don't want to create one, you can always ignore this step.

Once you create your API key, you will need to export that as:

```
export TAVILY_API_KEY="..."
```

```
from langchain_community.tools.tavily_search  
import TavilySearchResults
```

```
search = TavilySearchResults()
```

```
search.invoke("what is the weather in SF")
```

```
[{'url':
'https://www.metoffice.gov.uk/weather/forecast/'
'content': 'Thu 11 Jan Thu 11 Jan Seven day f
San Francisco San Francisco (United States of
weather Find a forecast Sat 6 Jan Sat 6 Jan Su
7 Jan Mon 8 Jan Mon 8 Jan Tue 9 Jan Tue 9 Jan W
Wed 10 Jan Thu 11 Jan Find a forecast Please c
location from the nearest places to : Forecast
Today Sat 6 Jan Sat 6 JanSan Francisco 7 day we
forecast including weather warnings, temperatur
wind, visibility, humidity and UV ... (11 Janua
Time 00:00 01:00 02:00 03:00 04:00 05:00 06:00
09:00 10:00 11:00 12:00 ... Oakland Int. 11.5 m
Francisco International 11.5 miles; Corte Mader
miles; Redwood City 23.4 miles;'},
{'url': 'https://www.latimes.com/travel/story/
11/east-brother-light-station-lighthouse-califo
'content': "May 18, 2023 Jan. 4, 2024 Subscr
unlimited accessSite Map Follow Us MORE FROM TH
TIMES Jan. 8, 2024 Travel & Experiences This m
Elysian Valley (a.k.a. Frogtown) Jan. 5, 2024 F
30, 2023The East Brother Light Station in
Francisco Bay is not a destination for eve
```


```
11, 2024 3 AM PT ... Champagne and hors d'oeuvre  
served in late afternoon – outdoors if ..."}]
```

## Retriever

We will also create a retriever over some data of our own. For a deeper explanation of each step here, see [this section](#)

```
from langchain.text_splitter import  
RecursiveCharacterTextSplitter  
from langchain_community.document_loaders import  
WebBaseLoader  
from langchain_community.vectorstores import FAISS  
from langchain_openai import OpenAIEmbeddings  
  
loader =  
WebBaseLoader("https://docs.smith.langchain.com")  
docs = loader.load()  
documents = RecursiveCharacterTextSplitter(  
    chunk_size=1000, chunk_overlap=200  
) .split_documents(docs)  
vector = FAISS.from_documents(documents,  
OpenAIEmbeddings())  
retriever = vector.as_retriever()
```

```
retriever.get_relevant_documents("how to  
upload a dataset")[0]
```

Document(page\_content="dataset uploading. Once we have a dataset, how can we use it to test changes to a prompt or chain? The most basic approach is to run the chain over the data points and visualize the outputs. Despite technological advancements, there still is no substitute for looking at outputs by eye. Currently, running the chain over the data points needs to be done client-side. The LangSmith client makes it easy to pull down a dataset and then run a chain over them, logging the results to a new project associated with the dataset. From there, you can review them. We've made it easy to assign feedback to runs and mark them as correct or incorrect directly in the web app, displaying aggregate statistics for each test project. We also make it easier to evaluate these runs. To that end, we've added a set of evaluators to the open-source LangChain library. These evaluators can be specified when initiating a test run and will evaluate the results once the test run completes. If we're being honest, most of", metadata={'source': 'https://docs.smith.langchain.com/overview', 'title': 'LangSmith Overview and User Guide |  LangSmith', 'description': 'Building reliable LLM applications can be challenging. LangChain simplifies the initial setup, but there is still work needed to bring the

```
performance of prompts, chains and agents up  
the level where they are reliable enough to  
be used in production.', 'language': 'en'})
```

Now that we have populated our index that we will do doing retrieval over, we can easily turn it into a tool (the format needed for an agent to properly use it)

```
from langchain.tools.retriever import  
create_retriever_tool
```

```
retriever_tool = create_retriever_tool(  
    retriever,  
    "langsmith_search",  
    "Search for information about LangSmith.  
    For any questions about LangSmith, you must  
    use this tool!",  
)
```

## Tools

Now that we have created both, we can create a list of tools that we will use downstream.

```
tools = [search, retriever_tool]
```

# Create the agent

Now that we have defined the tools, we can create the agent. We will be using an OpenAI Functions agent - for more information on this type of agent, as well as other options, see [this guide](#)

First, we choose the LLM we want to be guiding the agent.

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model="gpt-3.5-turbo",
temperature=0)
```

Next, we choose the prompt we want to use to guide the agent.

If you want to see the contents of this prompt and have access to LangSmith, you can go to:

<https://smith.langchain.com/hub/hwchase17/openai-functions-agent>

```
from langchain import hub

# Get the prompt to use - you can modify
```

```
this!
prompt = hub.pull("hwchase17/openai-
functions-agent")
prompt.messages
```

```
[SystemMessagePromptTemplate(prompt=PromptTemplate(
[], template='You are a helpful assistant')),
MessagesPlaceholder(variable_name='chat_history'),
HumanMessagePromptTemplate(prompt=PromptTemplate(
['input'], template='{input}')),
MessagesPlaceholder(variable_name='agent_scratchpad')]
```

Now, we can initialize the agent with the LLM, the prompt, and the tools. The agent is responsible for taking in input and deciding what actions to take. Crucially, the Agent does not execute those actions - that is done by the AgentExecutor (next step). For more information about how to think about these components, see our [conceptual guide](#)

```
from langchain.agents import
create_openai_functions_agent

agent = create_openai_functions_agent(llm,
tools, prompt)
```



Finally, we combine the agent (the brains) with the tools inside the AgentExecutor (which will repeatedly call the agent and execute tools). For more information about how to think about these components, see our [conceptual guide](#)

```
from langchain.agents import AgentExecutor

agent_executor = AgentExecutor(agent=agent,
                                tools=tools, verbose=True)
```

## Run the agent

We can now run the agent on a few queries! Note that for now, these are all **stateless** queries (it won't remember previous interactions).

```
agent_executor.invoke({"input": "hi!"})
```

```
> Entering new AgentExecutor chain...
Hello! How can I assist you today?






> Finished chain.
```


```
{'input': 'hi!', 'output': 'Hello! How can I assist you today?'}
```

```
agent_executor.invoke({"input": "how can langsmith help with testing?"})
```

> Entering new AgentExecutor chain...

Invoking: `langsmith\_search` with `{ 'query': 'LangSmith testing' }`

```
[Document(page_content='LangSmith Overview and User Guide |   LangSmith', metadata={'source': 'https://docs.smith.langchain.com/overview', 'title': 'LangSmith Overview and User Guide |   LangSmith', 'description': 'Building reliable LLM applications can be challenging. LangChain simplifies the initial setup, but there is still work needed to bring the performance of prompts, chains and agents up the level where they are reliable enough to be used in production.', 'language': 'en'}, Document(page_content='Skip to main content |  LangSmith DocsPython DocsJS/TS
```

DocsSearchGo to  
AppLangSmithOverviewTracingTesting &  
EvaluationOrganizationsHubLangSmith  
CookbookOverviewOn this pageLangSmith  
Overview and User GuideBuilding reliable LLM  
applications can be challenging. LangChain  
simplifies the initial setup, but there is  
still work needed to bring the performance of  
prompts, chains and agents up the level where  
they are reliable enough to be used in  
production.Over the past two months, we at  
LangChain have been building and using  
LangSmith with the goal of bridging this gap.  
This is our tactical user guide to outline  
effective ways to use LangSmith and maximize  
its benefits.On by default\u200bAt LangChain,  
all of us have LangSmith's tracing running in  
the background by default. On the Python  
side, this is achieved by setting environment  
variables, which we establish whenever we  
launch a virtual environment or open our bash  
shell and leave them set. The same principle  
applies to most JavaScript', metadata=  
{'source':  
'https://docs.smith.langchain.com/overview',  
'title': 'LangSmith Overview and User Guide |  
 LangSmith', 'description': 'Building  
reliable LLM applications can be challenging.  
LangChain simplifies the initial setup, but  
there is still work needed to bring the  
performance of prompts, chains and agents .

the level where they are reliable enough to be used in production.', 'language': 'en'}), Document(page\_content='You can also quickly edit examples and add them to datasets to expand the surface area of your evaluation sets or to fine-tune a model for improved quality or reduced costs. Monitoring\u200bAfter all this, your app might finally ready to go in production. LangSmith can also be used to monitor your application in much the same way that you used for debugging. You can log all traces, visualize latency and token usage statistics, and troubleshoot specific issues as they arise. Each run can also be assigned string tags or key-value metadata, allowing you to attach correlation ids or AB test variants, and filter runs accordingly. We've also made it possible to associate feedback programmatically with runs. This means that if your application has a thumbs up/down button on it, you can use that to log feedback back to LangSmith. This can be used to track performance over time and pinpoint under performing data points, which you can subsequently add to a dataset for future testing – mirroring the', metadata={'source': 'https://docs.smith.langchain.com/overview', 'title': 'LangSmith Overview and User Guide', 'description': 'Building reliable LLM applications can be challeng: \_

LangChain simplifies the initial setup, but there is still work needed to bring the performance of prompts, chains and agents up to the level where they are reliable enough to be used in production.', 'language': 'en'}), Document(page\_content='inputs, and see what happens. At some point though, our application is performing\nwell and we want to be more rigorous about testing changes. We can use a dataset\nthat we've constructed along the way (see above). Alternatively, we could spend some\ntime constructing a small dataset by hand. For these situations, LangSmith simplifies', metadata={'source': 'https://docs.smith.langchain.com/overview', 'title': 'LangSmith Overview and User Guide |  LangSmith', 'description': 'Building reliable LLM applications can be challenging. LangChain simplifies the initial setup, but there is still work needed to bring the performance of prompts, chains and agents up to the level where they are reliable enough to be used in production.', 'language': 'en'}})]LangSmith can help with testing in several ways. Here are some ways LangSmith can assist with testing:

1. Tracing: LangSmith provides tracing capabilities that can be used to monitor and debug your application during testing. You can log all traces, visualize latency and

token usage statistics, and troubleshoot specific issues as they arise.

2. Evaluation: LangSmith allows you to quickly edit examples and add them to datasets to expand the surface area of your evaluation sets. This can help you test and fine-tune your models for improved quality or reduced costs.

3. Monitoring: Once your application is ready for production, LangSmith can be used to monitor your application. You can log feedback programmatically with runs, track performance over time, and pinpoint underperforming data points. This information can be used to improve your application and add to datasets for future testing.

4. Rigorous Testing: When your application is performing well and you want to be more rigorous about testing changes, LangSmith can simplify the process. You can use existing datasets or construct small datasets by hand to test different scenarios and evaluate the performance of your application.

For more detailed information on how to use LangSmith for testing, you can refer to the [LangSmith Overview and User Guide] (<https://docs.smith.langchain.com/overview> ,

> Finished chain.

```
{'input': 'how can langsmith help with testing?',  
  'output': 'LangSmith can help with testing in several ways. Here are some ways LangSmith can assist with testing:\n\n1. Tracing: LangSmith provides tracing capabilities that can be used to monitor and debug your application during testing. You can log all traces, visualize latency and token usage statistics, and troubleshoot specific issues as they arise.\n\n2. Evaluation: LangSmith allows you to quickly edit examples and add them to datasets to expand the surface area of your evaluation sets. This can help you test and fine-tune your models for improved quality or reduced costs.\n\n3. Monitoring: Once your application is ready for production, LangSmith can be used to monitor your application. You can log feedback programmatically with runs, track performance over time, and pinpoint underperforming data points. This information can be used to improve your application and add to datasets for future testing.\n\n4. Rigorous Testing: When your application is performing well and you want to be more rigorous about testing changes, LangSmith simplify the process. You can use existing
```

```
datasets or construct small datasets by hand
to test different scenarios and evaluate the
performance of your application.\n\nFor more
detailed information on how to use LangSmith
for testing, you can refer to the [LangSmith
Overview and User Guide]
(https://docs.smith.langchain.com/overview).'} }
```

```
agent_executor.invoke({"input": "whats the
weather in sf?"})
```

> Entering new AgentExecutor chain...

Invoking: `tavily\_search\_results\_json` with  
`{'query': 'weather in San Francisco'}`

```
[{'url':
'https://www.whereandwhen.net/when/north-
america/california/san-francisco-
ca/january/', 'content': 'Best time to go to
San Francisco? Weather in San Francisco in
january 2024 How was the weather last
january? Here is the day by day recorded
weather in San Francisco in january 2023:
Seasonal average climate and temperature (
```



San Francisco in january 8% 46% 29% 12% 8%  
Evolution of daily average temperature and precipitation in San Francisco in january

Weather in San Francisco in january 2024. The weather in San Francisco in january comes from statistical datas on the past years. You can view the weather statistics the entire month, but also by using the tabs for the beginning, the middle and the end of the month. ... 11-01-2023 50°F to 54°F. 12-01-2023 50°F to 59°F. 13-01-2023 54°F to ...'}, {'url': 'https://www.latimes.com/travel/story/2024-01-11/east-brother-light-station-lighthouse-california', 'content': "May 18, 2023 Jan. 4, 2024 Subscribe for unlimited accessSite Map Follow Us MORE FROM THE L.A. TIMES Jan. 8, 2024 Travel & Experiences This must be Elysian Valley (a.k.a. Frogtown) Jan. 5, 2024 Food June 30, 2023The East Brother Light Station in the San Francisco Bay is not a destination for everyone. ... Jan. 11, 2024 3 AM PT ... Champagne and hors d'oeuvres are served in late afternoon – outdoors if ..."}]

I'm sorry, I couldn't find the current weather in San Francisco. However, you can check the weather in San Francisco by visiting a reliable weather website or using a weather app on your phone.

```
> Finished chain.
```

```
{'input': 'whats the weather in sf?',  
 'output': "I'm sorry, I couldn't find the  
current weather in San Francisco. However,  
you can check the weather in San Francisco by  
visiting a reliable weather website or using  
a weather app on your phone."}
```

## Adding in memory

As mentioned earlier, this agent is stateless. This means it does not remember previous interactions. To give it memory we need to pass in previous `chat_history`. Note: it needs to be called `chat_history` because of the prompt we are using. If we use a different prompt, we could change the variable name

```
# Here we pass in an empty list of messages  
for chat_history because it is the first  
message in the chat  
agent_executor.invoke({"input": "hi! my name  
is bob", "chat_history": []})
```

```
> Entering new AgentExecutor chain...  
Hello Bob! How can I assist you today?  
  
> Finished chain.
```

```
{'input': 'hi! my name is bob',  
 'chat_history': [],  
 'output': 'Hello Bob! How can I assist you  
today?'}
```

```
from langchain_core.messages import  
AIMessage, HumanMessage
```

```
agent_executor.invoke(  
    {  
        "chat_history": [  
            HumanMessage(content="hi! my name  
is bob"),  
            AIMessage(content="Hello Bob! How  
can I assist you today?"),  
        ],  
        "input": "what's my name?",  
    }  
)
```

```
> Entering new AgentExecutor chain...  
Your name is Bob.  
  
> Finished chain.
```

```
{'chat_history': [HumanMessage(content='hi!  
my name is bob'),  
    AIMessage(content='Hello Bob! How can I  
assist you today?')],  
 'input': "what's my name?",  
 'output': 'Your name is Bob.'}
```

If we want to keep track of these messages automatically, we can wrap this in a `RunnableWithMessageHistory`. For more information on how to use this, see [this guide](#)

```
from  
langchain_community.chat_message_histories  
import ChatMessageHistory  
from langchain_core.runnables.history import  
RunnableWithMessageHistory
```

```
message_history = ChatMessageHistory()
```

```
agent_with_chat_history =  
RunnableWithMessageHistory(  
    agent_executor,  
    # This is needed because in most real  
world scenarios, a session id is needed  
    # It isn't really used here because we  
are using a simple in memory  
ChatMessageHistory  
    lambda session_id: message_history,  
    input_messages_key="input",  
    history_messages_key="chat_history",  
)
```

```
agent_with_chat_history.invoke(  
    {"input": "hi! I'm bob"},  
    # This is needed because in most real  
world scenarios, a session id is needed  
    # It isn't really used here because we  
are using a simple in memory  
ChatMessageHistory  
    config={"configurable": {"session_id": "  
<foo>"}}},  
)
```

```
> Entering new AgentExecutor chain...  
Hello Bob! How can I assist you today?
```

```
> Finished chain.
```

```
{'input': "hi! I'm bob",  
  'chat_history': [],  
  'output': 'Hello Bob! How can I assist you  
today?'}
```

```
agent_with_chat_history.invoke(  
    {"input": "what's my name?"},  
    # This is needed because in most real  
world scenarios, a session id is needed  
    # It isn't really used here because we  
are using a simple in memory  
ChatMessageHistory  
    config={"configurable": {"session_id": "  
<foo>"}},  
)
```

```
> Entering new AgentExecutor chain...  
Your name is Bob.  
  
> Finished chain.
```

```
{'input': "what's my name?",  
 'chat_history': [HumanMessage(content="hi!  
I'm bob"),  
   AIMessage(content='Hello Bob! How can I  
assist you today?')],  
 'output': 'Your name is Bob.'}
```

## Conclusion

That's a wrap! In this quick start we covered how to create a simple agent. Agents are a complex topic, and there's a lot to learn! Head back to the [main agent page](#) to find more resources on conceptual guides, different types of agents, how to create custom tools, and more!