```
pip install langchain langchain-core langchain-community langgraph
langchain-huggingface transformers torch

Requirement already satisfied: langchain in
/usr/local/lib/python3.11/dist-packages (0.3.23)
Requirement already satisfied: langchain-core in
/usr/local/lib/python3.11/dist-packages (0.3.51)
Requirement already satisfied: langchain-community in
/usr/local/lib/python3.11/dist-packages (0.3.21)
Collecting langgraph
  Downloading langgraph-0.3.25-py3-none-any.whl.metadata (7.7 kB)
Collecting langchain-huggingface
  Downloading langchain_huggingface-0.1.2-py3-none-any.whl.metadata
(1.3 kB)
Requirement already satisfied: transformers in
/usr/local/lib/python3.11/dist-packages (4.50.3)
Requirement already satisfied: torch in
/usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: langchain-text-splitters<1.0.0,>=0.3.8
in /usr/local/lib/python3.11/dist-packages (from langchain) (0.3.8)
Requirement already satisfied: langsmith<0.4,>=0.1.17 in
/usr/local/lib/python3.11/dist-packages (from langchain) (0.3.22)
Requirement already satisfied: pydantic<3.0.0,>=2.7.4 in
/usr/local/lib/python3.11/dist-packages (from langchain) (2.11.1)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in
/usr/local/lib/python3.11/dist-packages (from langchain) (2.0.40)
Requirement already satisfied: requests<3,>=2 in
/usr/local/lib/python3.11/dist-packages (from langchain) (2.32.3)
Requirement already satisfied: PyYAML>=5.3 in
/usr/local/lib/python3.11/dist-packages (from langchain) (6.0.2)
Requirement already satisfied: tenacity!=8.4.0,<10.0.0,>=8.1.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-core) (9.1.2)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in
/usr/local/lib/python3.11/dist-packages (from langchain-core) (1.33)
Requirement already satisfied: packaging<25,>=23.2 in
/usr/local/lib/python3.11/dist-packages (from langchain-core) (24.2)
Requirement already satisfied: typing-extensions>=4.7 in
/usr/local/lib/python3.11/dist-packages (from langchain-core) (4.13.0)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(3.11.15)
Requirement already satisfied: dataclasses-json<0.7,>=0.5.7 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(0.6.7)
Requirement already satisfied: pydantic-settings<3.0.0,>=2.4.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(2.8.1)
Requirement already satisfied: httpx-sse<1.0.0,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(0.4.0)
```

```
Requirement already satisfied: numpy<3,>=1.26.2 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(1.26.4)
Collecting langgraph-checkpoint<3.0.0,>=2.0.10 (from langgraph)
  Downloading langgraph_checkpoint-2.0.24-py3-none-any.whl.metadata
(4.6 kB)
Collecting langgraph-prebuilt<0.2,>=0.1.1 (from langgraph)
  Downloading langgraph_prebuilt-0.1.8-py3-none-any.whl.metadata (5.0
kB)
Collecting langgraph-sdk<0.2.0,>=0.1.42 (from langgraph)
  Downloading langgraph_sdk-0.1.61-py3-none-any.whl.metadata (1.8 kB)
Collecting xxhash<4.0.0,>=3.5.0 (from langgraph)
  Downloading xxhash-3.5.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Requirement already satisfied: huggingface-hub>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-huggingface)
(0.30.1)
Requirement already satisfied: sentence-transformers>=2.6.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-huggingface)
(3.4.1)
Requirement already satisfied: tokenizers>=0.19.1 in
/usr/local/lib/python3.11/dist-packages (from langchain-huggingface)
(0.21.1)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.11/dist-packages (from transformers)
(2024.11.6)
Requirement already satisfied: safetensors>=0.4.3 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: networkx in
/usr/local/lib/python3.11/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.11/dist-packages (from torch) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127
in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in
/usr/local/lib/python3.11/dist-packages (from torch) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in
```

```
/usr/local/lib/python3.11/dist-packages (from torch) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in
/usr/local/lib/python3.11/dist-packages (from torch) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in
/usr/local/lib/python3.11/dist-packages (from torch) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in
/usr/local/lib/python3.11/dist-packages (from torch) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in
/usr/local/lib/python3.11/dist-packages (from torch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: triton==3.2.0 in
/usr/local/lib/python3.11/dist-packages (from torch) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in
/usr/local/lib/python3.11/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch)
(1.3.0)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (6.3.1)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (1.18.3)
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in
/usr/local/lib/python3.11/dist-packages (from dataclasses-
json<0.7,>=0.5.7->langchain-community) (3.26.1)
Requirement already satisfied: typing-inspect<1,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from dataclasses-
json<0.7,>=0.5.7->langchain-community) (0.9.0)
```

```
Requirement already satisfied: jsonpointer>=1.9 in
/usr/local/lib/python3.11/dist-packages (from jsonpatch<2.0,>=1.33-
>langchain-core) (3.0.0)
Collecting ormsgpack<2.0.0,>=1.8.0 (from langgraph-
checkpoint<3.0.0,>=2.0.10->langgraph)
  Downloading ormsgpack-1.9.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (43 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.5/43.5 kB 1.7 MB/s eta
0:00:00
ent already satisfied: httpx>=0.25.2 in
/usr/local/lib/python3.11/dist-packages (from langgraph-
sdk<0.2.0,>=0.1.42->langgraph) (0.28.1)
Requirement already satisfied: orjson>=3.10.1 in
/usr/local/lib/python3.11/dist-packages (from langgraph-
sdk<0.2.0,>=0.1.42->langgraph) (3.10.16)
Requirement already satisfied: requests-toolbelt<2.0.0,>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.17-
>langchain) (1.0.0)
Requirement already satisfied: zstandard<0.24.0,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.17-
>langchain) (0.23.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4-
>langchain) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4-
>langchain) (2.33.0)
Requirement already satisfied: typing-inspection>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4-
>langchain) (0.4.0)
Requirement already satisfied: python-dotenv>=0.21.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic-
settings<3.0.0,>=2.4.0->langchain-community) (1.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2-
>langchain) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2-
>langchain) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2-
>langchain) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2-
>langchain) (2025.1.31)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.11/dist-packages (from sentence-
transformers>=2.6.0->langchain-huggingface) (1.6.1)
Requirement already satisfied: scipy in
```

```
/usr/local/lib/python3.11/dist-packages (from sentence-
transformers>=2.6.0->langchain-huggingface) (1.14.1)
Requirement already satisfied: Pillow in
/usr/local/lib/python3.11/dist-packages (from sentence-
transformers>=2.6.0->langchain-huggingface) (11.1.0)
Requirement already satisfied: greenlet>=1 in
/usr/local/lib/python3.11/dist-packages (from SQLAlchemy<3,>=1.4-
>langchain) (3.1.1)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->torch) (3.0.2)
Requirement already satisfied: anyio in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.25.2-
>langgraph-sdk<0.2.0,>=0.1.42->langgraph) (4.9.0)
Requirement already satisfied: httpcore==1.* in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.25.2-
>langgraph-sdk<0.2.0,>=0.1.42->langgraph) (1.0.7)
Requirement already satisfied: h11<0.15,>=0.13 in
/usr/local/lib/python3.11/dist-packages (from httpcore==1.*-
>httpx>=0.25.2->langgraph-sdk<0.2.0,>=0.1.42->langgraph) (0.14.0)
Requirement already satisfied: mypy-extensions>=0.3.0 in
/usr/local/lib/python3.11/dist-packages (from typing-
inspect<1,>=0.4.0->dataclasses-json<0.7,>=0.5.7->langchain-community)
(1.0.0)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence-
transformers>=2.6.0->langchain-huggingface) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence-
transformers>=2.6.0->langchain-huggingface) (3.6.0)
Requirement already satisfied: sniffio>=1.1 in
/usr/local/lib/python3.11/dist-packages (from anyio->httpx>=0.25.2-
>langgraph-sdk<0.2.0,>=0.1.42->langgraph) (1.3.1)
Downloading langgraph-0.3.25-py3-none-any.whl (142 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 142.4/142.4 kB 10.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.0/42.0 kB 2.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.2/47.2 kB 2.0 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 13.7 MB/s eta
0:00:00
sgpack-1.9.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (223 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 223.6/223.6 kB 19.4 MB/s eta
0:00:00
sgpack, langgraph-sdk, langgraph-checkpoint, langgraph-prebuilt,
langchain-huggingface, langgraph
Successfully installed langchain-huggingface-0.1.2 langgraph-0.3.25
```

```
langgraph-checkpoint-2.0.24 langgraph-prebuilt-0.1.8 langgraph-sdk-
0.1.61 ormsgpack-1.9.1 xxhash-3.5.0

pip install unstructured

Collecting unstructured
  Downloading unstructured-0.17.2-py3-none-any.whl.metadata (24 kB)
Requirement already satisfied: chardet in
/usr/local/lib/python3.11/dist-packages (from unstructured) (5.2.0)
Collecting filetype (from unstructured)
  Downloading filetype-1.2.0-py2.py3-none-any.whl.metadata (6.5 kB)
Collecting python-magic (from unstructured)
  Downloading python_magic-0.4.27-py2.py3-none-any.whl.metadata (5.8
kB)
Requirement already satisfied: lxml in /usr/local/lib/python3.11/dist-
packages (from unstructured) (5.3.1)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-
packages (from unstructured) (3.9.1)
Requirement already satisfied: requests in
/usr/local/lib/python3.11/dist-packages (from unstructured) (2.32.3)
Requirement already satisfied: beautifulsoup4 in
/usr/local/lib/python3.11/dist-packages (from unstructured) (4.13.3)
Collecting emoji (from unstructured)
  Downloading emoji-2.14.1-py3-none-any.whl.metadata (5.7 kB)
Requirement already satisfied: dataclasses-json in
/usr/local/lib/python3.11/dist-packages (from unstructured) (0.6.7)
Collecting python-iso639 (from unstructured)
  Downloading python_iso639-2025.2.18-py3-none-any.whl.metadata (14
kB)
Collecting langdetect (from unstructured)
  Downloading langdetect-1.0.9.tar.gz (981 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 981.5/981.5 kB 16.1 MB/s eta
0:00:00
etadata (setup.py) ... ent already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from unstructured) (1.26.4)
Collecting rapidfuzz (from unstructured)
  Downloading rapidfuzz-3.13.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Requirement already satisfied: backoff in
/usr/local/lib/python3.11/dist-packages (from unstructured) (2.2.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.11/dist-packages (from unstructured) (4.13.0)
Collecting unstructured-client (from unstructured)
  Downloading unstructured_client-0.32.1-py3-none-any.whl.metadata (22
kB)
Requirement already satisfied: wrapt in
/usr/local/lib/python3.11/dist-packages (from unstructured) (1.17.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from unstructured) (4.67.1)
Requirement already satisfied: psutil in
```

```
/usr/local/lib/python3.11/dist-packages (from unstructured) (5.9.5)
Collecting python-oxmsg (from unstructured)
  Downloading python_oxmsg-0.0.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: html5lib in
/usr/local/lib/python3.11/dist-packages (from unstructured) (1.1)
Requirement already satisfied: soupsieve>1.2 in
/usr/local/lib/python3.11/dist-packages (from beautifulsoup4-
>unstructured) (2.6)
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in
/usr/local/lib/python3.11/dist-packages (from dataclasses-json-
>unstructured) (3.26.1)
Requirement already satisfied: typing-inspect<1,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from dataclasses-json-
>unstructured) (0.9.0)
Requirement already satisfied: six>=1.9 in
/usr/local/lib/python3.11/dist-packages (from html5lib->unstructured)
(1.17.0)
Requirement already satisfied: webencodings in
/usr/local/lib/python3.11/dist-packages (from html5lib->unstructured)
(0.5.1)
Requirement already satisfied: click in
/usr/local/lib/python3.11/dist-packages (from nltk->unstructured)
(8.1.8)
Requirement already satisfied: joblib in
/usr/local/lib/python3.11/dist-packages (from nltk->unstructured)
(1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk->unstructured)
(2024.11.6)
Collecting olefile (from python-oxmsg->unstructured)
  Downloading olefile-0.47-py2.py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->unstructured)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests->unstructured)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->unstructured)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->unstructured)
(2025.1.31)
Collecting aiofiles>=24.1.0 (from unstructured-client->unstructured)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: cryptography>=3.1 in
/usr/local/lib/python3.11/dist-packages (from unstructured-client-
>unstructured) (43.0.3)
Collecting eval-type-backport>=0.2.0 (from unstructured-client-
```

```
>unstructured)
  Downloading eval_type_backport-0.2.2-py3-none-any.whl.metadata (2.2
kB)
Requirement already satisfied: httpx>=0.27.0 in
/usr/local/lib/python3.11/dist-packages (from unstructured-client-
>unstructured) (0.28.1)
Requirement already satisfied: nest-asyncio>=1.6.0 in
/usr/local/lib/python3.11/dist-packages (from unstructured-client-
>unstructured) (1.6.0)
Requirement already satisfied: pydantic>=2.10.3 in
/usr/local/lib/python3.11/dist-packages (from unstructured-client-
>unstructured) (2.11.1)
Collecting pypdf>=4.0 (from unstructured-client->unstructured)
  Downloading pypdf-5.4.0-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from unstructured-client-
>unstructured) (2.8.2)
Requirement already satisfied: requests-toolbelt>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from unstructured-client-
>unstructured) (1.0.0)
Requirement already satisfied: typing-inspection>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from unstructured-client-
>unstructured) (0.4.0)
Requirement already satisfied: cffi>=1.12 in
/usr/local/lib/python3.11/dist-packages (from cryptography>=3.1-
>unstructured-client->unstructured) (1.17.1)
Requirement already satisfied: anyio in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.27.0-
>unstructured-client->unstructured) (4.9.0)
Requirement already satisfied: httpcore==1.* in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.27.0-
>unstructured-client->unstructured) (1.0.7)
Requirement already satisfied: h11<0.15,>=0.13 in
/usr/local/lib/python3.11/dist-packages (from httpcore==1.*-
>httpx>=0.27.0->unstructured-client->unstructured) (0.14.0)
Requirement already satisfied: packaging>=17.0 in
/usr/local/lib/python3.11/dist-packages (from
marshmallow<4.0.0,>=3.18.0->dataclasses-json->unstructured) (24.2)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.10.3-
>unstructured-client->unstructured) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.10.3-
>unstructured-client->unstructured) (2.33.0)
Requirement already satisfied: mypy-extensions>=0.3.0 in
/usr/local/lib/python3.11/dist-packages (from typing-
inspect<1,>=0.4.0->dataclasses-json->unstructured) (1.0.0)
Requirement already satisfied: pycparser in
/usr/local/lib/python3.11/dist-packages (from cffi>=1.12-
```

```
>cryptography>=3.1->unstructured-client->unstructured) (2.22)
Requirement already satisfied: sniffio>=1.1 in
/usr/local/lib/python3.11/dist-packages (from anyio->httpx>=0.27.0-
>unstructured-client->unstructured) (1.3.1)
Downloading unstructured-0.17.2-py3-none-any.whl (1.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 46.7 MB/s eta
0:00:00
oji-2.14.1-py3-none-any.whl (590 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 590.6/590.6 kB 10.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 167.6/167.6 kB 15.1 MB/s eta
0:00:00
agic-0.4.27-py2.py3-none-any.whl (13 kB)
Downloading python_oxmsg-0.0.2-py3-none-any.whl (31 kB)
Downloading rapidfuzz-3.13.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 32.0 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 180.5/180.5 kB 12.8 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.3/302.3 kB 18.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.6/114.6 kB 10.2 MB/s eta
0:00:00
e=langdetect-1.0.9-py3-none-any.whl size=993223
sha256=2886234dd7e1c1818030f8af180c35b9e0490f0f8a2c6943aaf94bcc46c57d3
0
  Stored in directory:
/root/.cache/pip/wheels/0a/f2/b2/e5ca405801e05eb7c8ed5b3b4bcf1fcabcd62
72c167640072e
Successfully built langdetect
Installing collected packages: filetype, rapidfuzz, python-magic,
python-iso639, pypdf, olefile, langdetect, eval-type-backport, emoji,
aiofiles, python-oxmsg, unstructured-client, unstructured
Successfully installed aiofiles-24.1.0 emoji-2.14.1 eval-type-
backport-0.2.2 filetype-1.2.0 langdetect-1.0.9 olefile-0.47 pypdf-
5.4.0 python-iso639-2025.2.18 python-magic-0.4.27 python-oxmsg-0.0.2
rapidfuzz-3.13.0 unstructured-0.17.2 unstructured-client-0.32.1
```

```
from langchain_community.document_loaders import UnstructuredURLLoader
urls={'https://langchain-ai.github.io/langgraph/tutorials/introduction
/'}
loader=UnstructuredURLLoader(urls=urls)
docs=loader.load()

docs[0]
```

```
Document(metadata={'source':
'https://langchain-ai.github.io/langgraph/tutorials/introduction/'},
page_content='□ LangGraph Quickstart¶\n\nIn this tutorial, we will
```

```
build a support chatbot in LangGraph that can:\n\n Answer common questions by searching the web  Maintain conversation state across calls  Route complex queries to a human for review  Use custom state to control its behavior  Rewind and explore alternative conversation paths\n\nWe\'ll start with a basic chatbot and progressively add more sophisticated capabilities, introducing key LangGraph concepts along the way. Let’s dive in! \n\nSetup¶\n\nFirst, install the required packages and configure your environment:\n\n%%capture --no-stderr\n%pip install -U langgraph langsmith langchain_anthropic\n\nimport getpass\nimport os\n\n\ndef _set_env(var: str):\n    if not os.environ.get(var):\n        os.environ[var] = getpass.getpass(f"{var}: ")\n\n\n_set_env("ANTHROPIC_API_KEY")\n\nSet up LangSmith for LangGraph development\n\nSign up for LangSmith to quickly spot issues and improve the performance of your LangGraph projects. LangSmith lets you use trace data to debug, test, and monitor your LLM apps built with LangGraph — read more about how to get started here.\n\nPart 1: Build a Basic Chatbot¶\n\nWe\'ll first create a simple chatbot using LangGraph. This chatbot will respond directly to user messages. Though simple, it will illustrate the core concepts of building with LangGraph. By the end of this section, you will have a built rudimentary chatbot.\n\nStart by creating a StateGraph. A StateGraph object defines the structure of our chatbot as a "state machine". We\'ll add nodes to represent the llm and functions our chatbot can call and edges to specify how the bot should transition between these functions.\n\nfrom typing import Annotated\n\nfrom typing_extensions import TypedDict\n\nfrom langgraph.graph import StateGraph, START, END\nfrom langgraph.graph.message import add_messages\n\n\nclass State(TypedDict):\n    # Messages have the type "list". The `add_messages` function\n    # in the annotation defines how this state key should be updated\n    # (in this case, it appends messages to the list, rather than overwriting them)\n    messages: Annotated[list, add_messages]\n\n\ngraph_builder = StateGraph(State)\n\nAPI Reference: StateGraph | START | END | add_messages\n\nOur graph can now handle two key tasks:\n\nEach node can receive the current State as input and output an update to the state.\n\nUpdates to messages will be appended to the existing list rather than overwriting it, thanks to the prebuilt add_messages function used with the Annotated syntax.\n\nConcept\n\nWhen defining a graph, the first step is to define its State. The State includes the graph\'s schema and reducer functions that handle state updates. In our example, State is a TypedDict with one key: messages. The add_messages reducer function is used to append new messages to the list instead of overwriting it. Keys without a reducer annotation will overwrite previous values. Learn more about state, reducers, and related concepts in this guide.\n\nNext, add a "chatbot" node. Nodes represent units of work. They are typically regular python functions.\n\n\nfrom langchain_anthropic import ChatAnthropic\n\nllm = ChatAnthropic(model="claude-3-5-sonnet-20240620")\n\n\ndef chatbot(state: State):\n    return {"messages":
```

```
[llm.invoke(state["messages"])]}\n\n\n# The first argument is the unique node name\n# The second argument is the function or object that will be called whenever\n# the node is used.\ngraph_builder.add_node("chatbot", chatbot)\n\nAPI Reference: ChatAnthropic\n\nNotice how the chatbot node function takes the current State as input and returns a dictionary containing an updated messages list under the key "messages". This is the basic pattern for all LangGraph node functions.\n\nThe add_messages function in our State will append the llm\'s response messages to whatever messages are already in the state.\n\nNext, add an entry point. This tells our graph where to start its work each time we run it.\n\ngraph_builder.add_edge(START, "chatbot")\n\nSimilarly, set a finish point. This instructs the graph "any time this node is run, you can exit."\n\ngraph_builder.add_edge("chatbot", END)\n\nFinally, we\'ll want to be able to run our graph. To do so, call "compile()" on the graph builder. This creates a "CompiledGraph" we can use invoke on our state.\n\ngraph = graph_builder.compile()\n\nYou can visualize the graph using the get_graph method and one of the "draw" methods, like draw_ascii or draw_png. The draw methods each require additional dependencies.\n\nfrom IPython.display import Image, display\n\ntry:\n    display(Image(graph.get_graph().draw_mermaid_png()))\nexcept Exception:\n    # This requires some extra dependencies and is optional\n    pass\n\n\n\nNow let\'s run the chatbot!\n\nTip: You can exit the chat loop at any time by typing "quit", "exit", or "q".\n\ndef stream_graph_updates(user_input: str):\n    for event in graph.stream({"messages": [{"role": "user", "content": user_input}]}):\n        for value in event.values():\n            print("Assistant:", value["messages"][-1].content)\n\n\nwhile True:\n    try:\n        user_input = input("User: ")\n        if user_input.lower() in ["quit", "exit", "q"]:\n            print("Goodbye!")\n            break\n        stream_graph_updates(user_input)\n    except:\n        # fallback if input() is not available\n        user_input = "What do you know about LangGraph?"\n        print("User: " + user_input)\n        stream_graph_updates(user_input)\n        break\n\nAssistant: LangGraph is a library designed to help build stateful multi-agent applications using language models. It provides tools for creating workflows and state machines to coordinate multiple AI agents or language model interactions. LangGraph is built on top of LangChain, leveraging its components while adding graph-based coordination capabilities. It\'s particularly useful for developing more complex, stateful AI applications that go beyond simple query-response interactions.\nGoodbye!\n\nCongratulations! You\'ve built your first chatbot using LangGraph. This bot can engage in basic conversation by taking user input and generating responses using an LLM. You can inspect a LangSmith Trace for the call above at the provided link.\n\nHowever, you may have noticed that the bot\'s knowledge is limited to what\'s in its training data. In the next part, we\'ll add a web search tool to expand the bot\'s knowledge and make it more capable.\
```

n\nBelow is the full code for this section for your reference:\n\nPart 2: Enhancing the Chatbot with Tools¶\n\nTo handle queries our chatbot can\'t answer "from memory", we\'ll integrate a web search tool. Our bot can use this tool to find relevant information and provide better responses.\n\nRequirements¶\n\nBefore we start, make sure you have the necessary packages installed and API keys set up:\n\nFirst, install the requirements to use the Tavily Search Engine, and set your TAVILY_API_KEY.\n\n%%capture --no-stderr\n%pip install -U tavily-python langchain_community\n\n_set_env("TAVILY_API_KEY")\n\nTAVILY_API_KEY: ·····\n\nNext, define the tool:\n\nfrom langchain_community.tools.tavily_search import TavilySearchResults\n\ntool = TavilySearchResults(max_results=2)\ntools = [tool]\ntool.invoke("What\'s a \'node\' in LangGraph?")\n\nAPI Reference: TavilySearchResults\n\n[{\'url\': \'https://medium.com/@cplog/introduction-to-langgraph-a-beginners-guide-14f9be027141\',\n \'content\': \'Nodes: Nodes are the building blocks of your LangGraph. Each node represents a function or a computation step. You define nodes to perform specific tasks, such as processing input, making ...\'},\n {\'url\': \'https://saksheepatil05.medium.com/demystifying-langgraph-a-beginner-friendly-dive-into-langgraph-concepts-5ffe890ddac0\',\n \'content\': \'Nodes (Tasks): Nodes are like the workstations on the assembly line. Each node performs a specific task on the product. In LangGraph, nodes are Python functions that take the current state, do some work, and return an updated state. Next, we define the nodes, each representing a task in our sandwich-making process.\'}]\n\nThe results are page summaries our chat bot can use to answer questions.\n\nNext, we\'ll start defining our graph. The following is all the same as in Part 1, except we have added bind_tools on our LLM. This lets the LLM know the correct JSON format to use if it wants to use our search engine.\n\nfrom typing import Annotated\n\nfrom langchain_anthropic import ChatAnthropic\nfrom typing_extensions import TypedDict\n\nfrom langgraph.graph import StateGraph, START, END\nfrom langgraph.graph.message import add_messages\n\n\nclass State(TypedDict):\n   messages: Annotated[list, add_messages]\n\n\ngraph_builder = StateGraph(State)\n\n\nllm = ChatAnthropic(model="claude-3-5-sonnet-20240620")\n# Modification: tell the LLM which tools it can call\nllm_with_tools = llm.bind_tools(tools)\n\n\ndef chatbot(state: State):\n   return {"messages": [llm_with_tools.invoke(state["messages"])]}\n\n\ngraph_builder.add_node("chatbot", chatbot)\n\nAPI Reference: ChatAnthropic | StateGraph | START | END | add_messages\n\nNext we need to create a function to actually run the tools if they are called. We\'ll do this by adding the tools to a new node.\n\nBelow, we implement a BasicToolNode that checks the most recent message in the state and calls tools if the message contains tool_calls. It relies on the LLM\'s tool_calling support, which is available in Anthropic, OpenAI, Google Gemini, and a number of other LLM providers.\n\nWe will later replace this with LangGraph\'s prebuilt ToolNode to speed things

```
up, but building it ourselves first is instructive.\n\nimport json\n\
nfrom langchain_core.messages import ToolMessage\n\n\nclass
BasicToolNode:\n    """A node that runs the tools requested in the
last AIMessage."""\n\n    def __init__(self, tools: list) -> None:\n
self.tools_by_name = {tool.name: tool for tool in tools}\n\n    def
__call__(self, inputs: dict):\n        if messages :=
inputs.get("messages", []):\n            message = messages[-1]\n
else:\n            raise ValueError("No message found in input")\n
outputs = []\n        for tool_call in message.tool_calls:\n
tool_result = self.tools_by_name[tool_call["name"]].invoke(\n
tool_call["args"]\n            )\n            outputs.append(\n
ToolMessage(\n                    content=json.dumps(tool_result),\n
name=tool_call["name"],\n
tool_call_id=tool_call["id"],\n                )\n            )\n
return {"messages": outputs}\n\n\ntool_node =
BasicToolNode(tools=[tool])\ngraph_builder.add_node("tools",
tool_node)\n\nAPI Reference: ToolMessage\n\nWith the tool node added,
we can define the conditional_edges.\n\nRecall that edges route the
control flow from one node to the next. Conditional edges usually
contain "if" statements to route to different nodes depending on the
current graph state. These functions receive the current graph state
and return a string or list of strings indicating which node(s) to
call next.\n\nBelow, call define a router function called route_tools,
that checks for tool_calls in the chatbot\'s output. Provide this
function to the graph by calling add_conditional_edges, which tells
the graph that whenever the chatbot node completes to check this
function to see where to go next.\n\nThe condition will route to tools
if tool calls are present and END if not.\n\nLater, we will replace
this with the prebuilt tools_condition to be more concise, but
implementing it ourselves first makes things more clear.\n\ndef
route_tools(\n    state: State,\n):\n    """\n    Use in the
conditional_edge to route to the ToolNode if the last message\n    has
tool calls. Otherwise, route to the end.\n    """\n    if
isinstance(state, list):\n        ai_message = state[-1]\n    elif
messages := state.get("messages", []):\n        ai_message =
messages[-1]\n    else:\n        raise ValueError(f"No messages found
in input state to tool_edge: {state}")\n    if hasattr(ai_message,
"tool_calls") and len(ai_message.tool_calls) > 0:\n        return
"tools"\n    return END\n\n\n# The `tools_condition` function returns
"tools" if the chatbot asks to use a tool, and "END" if\n# it is fine
directly responding. This conditional routing defines the main agent
loop.\ngraph_builder.add_conditional_edges(\n    "chatbot",\n
route_tools,\n    # The following dictionary lets you tell the graph
to interpret the condition\'s outputs as a specific node\n    # It
defaults to the identity function, but if you\n    # want to use a
node named something else apart from "tools",\n    # You can update
the value of the dictionary to something else\n    # e.g., "tools":
"my_tools"\n    {"tools": "tools", END: END},\n)\n# Any time a tool is
called, we return to the chatbot to decide the next step\
```

ngraph_builder.add_edge("tools", "chatbot")\
ngraph_builder.add_edge(START, "chatbot")\ngraph =
graph_builder.compile()\n\nNotice that conditional edges start from a
single node. This tells the graph "any time the \'chatbot\' node runs,
either go to \'tools\' if it calls a tool, or end the loop if it
responds directly.\n\nLike the prebuilt tools_condition, our function
returns the END string if no tool calls are made. When the graph
transitions to END, it has no more tasks to complete and ceases
execution. Because the condition can return END, we don\'t need to
explicitly set a finish_point this time. Our graph already has a way
to finish!\n\nLet\'s visualize the graph we\'ve built. The following
function has some additional dependencies to run that are unimportant
for this tutorial.\n\nfrom IPython.display import Image, display\n\
ntry:\n   display(Image(graph.get_graph().draw_mermaid_png()))\
nexcept Exception:\n   # This requires some extra dependencies and is
optional\n   pass\n\n\n\nNow we can ask the bot questions outside its
training data.\n\nwhile True:\n   try:\n   user_input =
input("User: ")\n   if user_input.lower() in ["quit", "exit",
"q"]:\n   print("Goodbye!")\n   break\n\n
stream_graph_updates(user_input)\n   except:\n   # fallback if
input() is not available\n   user_input = "What do you know about
LangGraph?"\n   print("User: " + user_input)\n
stream_graph_updates(user_input)\n   break\n\nAssistant:
[{\'text\': "To provide you with accurate and up-to-date information
about LangGraph, I\'ll need to search for the latest details. Let me
do that for you.", \'type\': \'text\'},
{\'id\': \'toolu_01Q588CszHaSvvP2MxRq9zRD\', \'input\':
{\'query\': \'LangGraph AI tool
information\'}, \'name\': \'tavily_search_results_json\', \'type\': \'
tool_use\'}]\nAssistant: [{"url":
"https://www.langchain.com/langgraph", "content": "LangGraph sets the
foundation for how we can build and scale AI workloads \\u2014 from
conversational agents, complex task automation, to custom LLM-backed
experiences that \'just work\'. The next chapter in building complex
production-ready features with LLMs is agentic, and with LangGraph and
LangSmith, LangChain delivers an out-of-the-box solution ..."},
{"url": "https://github.com/langchain-ai/langgraph", "content":
"Overview. LangGraph is a library for building stateful, multi-actor
applications with LLMs, used to create agent and multi-agent
workflows. Compared to other LLM frameworks, it offers these core
benefits: cycles, controllability, and persistence. LangGraph allows
you to define flows that involve cycles, essential for most agentic
architectures ..."}]\nAssistant: Based on the search results, I can
provide you with information about LangGraph:\n\n1. Purpose:\n
LangGraph is a library designed for building stateful, multi-actor
applications with Large Language Models (LLMs). It\'s particularly
useful for creating agent and multi-agent workflows.\n\n2. Developer:\
n   LangGraph is developed by LangChain, a company known for its tools
and frameworks in the AI and LLM space.\n\n3. Key Features:\n   -

Cycles: LangGraph allows the definition of flows that involve cycles, which is essential for most agentic architectures.\n   - Controllability: It offers enhanced control over the application flow.\n   - Persistence: The library provides ways to maintain state and persistence in LLM-based applications.\n\n4. Use Cases:\n   LangGraph can be used for various applications, including:\n   - Conversational agents\n   - Complex task automation\n   - Custom LLM-backed experiences\n\n5. Integration:\n   LangGraph works in conjunction with LangSmith, another tool by LangChain, to provide an out-of-the-box solution for building complex, production-ready features with LLMs.\n\n6. Significance:\n   LangGraph is described as setting the foundation for building and scaling AI workloads. It\'s positioned as a key tool in the next chapter of LLM-based application development, particularly in the realm of agentic AI.\n\n7. Availability:\n   LangGraph is open-source and available on GitHub, which suggests that developers can access and contribute to its codebase.\n\n8. Comparison to Other Frameworks:\n   LangGraph is noted to offer unique benefits compared to other LLM frameworks, particularly in its ability to handle cycles, provide controllability, and maintain persistence.\n\nLangGraph appears to be a significant tool in the evolving landscape of LLM-based application development, offering developers new ways to create more complex, stateful, and interactive AI systems.\nGoodbye!\n\nCongrats! You\'ve created a conversational agent in langgraph that can use a search engine to retrieve updated information when needed. Now it can handle a wider range of user queries. To inspect all the steps your agent just took, check out this LangSmith trace.\n\nOur chatbot still can\'t remember past interactions on its own, limiting its ability to have coherent, multi-turn conversations. In the next part, we\'ll add memory to address this.\n\nThe full code for the graph we\'ve created in this section is reproduced below, replacing our BasicToolNode for the prebuilt ToolNode, and our route_tools condition with the prebuilt tools_condition\n\nPart 3: Adding Memory to the Chatbot¶\n\nOur chatbot can now use tools to answer user questions, but it doesn\'t remember the context of previous interactions. This limits its ability to have coherent, multi-turn conversations.\n\nLangGraph solves this problem through persistent checkpointing. If you provide a checkpointer when compiling the graph and a thread_id when calling your graph, LangGraph automatically saves the state after each step. When you invoke the graph again using the same thread_id, the graph loads its saved state, allowing the chatbot to pick up where it left off.\n\nWe will see later that checkpointing is much more powerful than simple chat memory - it lets you save and resume complex state at any time for error recovery, human-in-the-loop workflows, time travel interactions, and more. But before we get too ahead of ourselves, let\'s add checkpointing to enable multi-turn conversations.\n\nTo get started, create a MemorySaver checkpointer.\n\nfrom langgraph.checkpoint.memory import MemorySaver\n\nmemory = MemorySaver()\n\nAPI Reference: MemorySaver\n\nNotice we\'re using an

```
in-memory checkpointer. This is convenient for our tutorial (it saves it all in-memory). In a production application, you would likely change this to use SqliteSaver or PostgresSaver and connect to your own DB.\n\nNext define the graph. Now that you\'ve already built your own BasicToolNode, we\'ll replace it with LangGraph\'s prebuilt ToolNode and tools_condition, since these do some nice things like parallel API execution. Apart from that, the following is all copied from Part 2.\n\nfrom typing import Annotated\n\nfrom langchain_anthropic import ChatAnthropic\nfrom langchain_community.tools.tavily_search import TavilySearchResults\nfrom langchain_core.messages import BaseMessage\nfrom typing_extensions import TypedDict\n\nfrom langgraph.graph import StateGraph, START, END\nfrom langgraph.graph.message import add_messages\nfrom langgraph.prebuilt import ToolNode, tools_condition\n\n\nclass State(TypedDict):\n    messages: Annotated[list, add_messages]\n\n\ngraph_builder = StateGraph(State)\n\n\ntool = TavilySearchResults(max_results=2)\ntools = [tool]\nllm = ChatAnthropic(model="claude-3-5-sonnet-20240620")\nllm_with_tools = llm.bind_tools(tools)\n\n\ndef chatbot(state: State):\n    return {"messages": [llm_with_tools.invoke(state["messages"])]}\n\n\ngraph_builder.add_node("chatbot", chatbot)\n\ntool_node = ToolNode(tools=[tool])\ngraph_builder.add_node("tools", tool_node)\n\ngraph_builder.add_conditional_edges(\n    "chatbot",\n    tools_condition,\n)\n# Any time a tool is called, we return to the chatbot to decide the next step\ngraph_builder.add_edge("tools", "chatbot")\ngraph_builder.add_edge(START, "chatbot")\n\nAPI Reference: ChatAnthropic | TavilySearchResults | BaseMessage | StateGraph | START | END | add_messages | ToolNode | tools_condition\n\nFinally, compile the graph with the provided checkpointer.\n\ngraph = graph_builder.compile(checkpointer=memory)\n\nNotice the connectivity of the graph hasn\'t changed since Part 2. All we are doing is checkpointing the State as the graph works through each node.\n\nfrom IPython.display import Image, display\n\ntry:\n    display(Image(graph.get_graph().draw_mermaid_png()))\nexcept Exception:\n    # This requires some extra dependencies and is optional\n    pass\n\n\n\nNow you can interact with your bot! First, pick a thread to use as the key for this conversation.\n\nconfig = {"configurable": {"thread_id": "1"}}\n\nNext, call your chat bot.\n\nuser_input = "Hi there! My name is Will."\n\n# The config is the **second positional argument** to stream() or invoke()!\nevents = graph.stream(\n    {"messages": [{"role": "user", "content": user_input}]},\n    config,\n    stream_mode="values",\n)\nfor event in events:\n    event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nHi there! My name is Will.\n==================================[1m Ai Message [0m==================================\n\nHello Will! It\'s nice to meet you. How can I assist you today? Is there anything specific you\'d like to know or discuss?\n\nNote: The config was provided as
```

```
the second positional argument when calling our graph. It importantly is not nested within the graph inputs ({\'messages\': []}).\n\nLet\'s ask a followup: see if it remembers your name.\n\nuser_input = "Remember my name?"\n\n# The config is the **second positional argument** to stream() or invoke()!\nevents = graph.stream(\n {"messages": [{"role": "user", "content": user_input}]},\n    config,\n    stream_mode="values",\n)\nfor event in events:\n event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nRemember my name?\n==================================[1m Ai Message [0m==================================\n\nOf course, I remember your name, Will. I always try to pay attention to important details that users share with me. Is there anything else you\'d like to talk about or any questions you have? I\'m here to help with a wide range of topics or tasks.\n\nNotice that we aren\'t using an external list for memory: it\'s all handled by the checkpointer! You can inspect the full execution in this LangSmith trace to see what\'s going on.\n\nDon\'t believe me? Try this using a different config.\n\n# The only difference is we change the `thread_id` here to "2" instead of "1"\nevents = graph.stream(\n    {"messages": [{"role": "user", "content": user_input}]},\n    {"configurable": {"thread_id": "2"}},\n stream_mode="values",\n)\nfor event in events:\n    event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nRemember my name?\n==================================[1m Ai Message [0m==================================\n\nI apologize, but I don\'t have any previous context or memory of your name. As an AI assistant, I don\'t retain information from past conversations. Each interaction starts fresh. Could you please tell me your name so I can address you properly in this conversation?\n\nNotice that the only change we\'ve made is to modify the thread_id in the config. See this call\'s LangSmith trace for comparison.\n\nBy now, we have made a few checkpoints across two different threads. But what goes into a checkpoint? To inspect a graph\'s state for a given config at any time, call get_state(config).\n\nsnapshot = graph.get_state(config)\nsnapshot\n\nStateSnapshot(values={\'messages\': [HumanMessage(content=\'Hi there! My name is Will.\', additional_kwargs={}, response_metadata={}, id=\'8c1ca919-c553-4ebf-95d4-b59a2d61e078\'), AIMessage(content="Hello Will! It\'s nice to meet you. How can I assist you today? Is there anything specific you\'d like to know or discuss?", additional_kwargs={}, response_metadata={\'id\': \'msg_01WTQebPhNwmMrmmWojJ9KXJ\', \'model\': \'claude-3-5-sonnet-20240620\', \'stop_reason\': \'end_turn\', \'stop_sequence\': None, \'usage\': {\'input_tokens\': 405, \'output_tokens\': 32}}, id=\'run-58587b77-8c82-41e6-8a90-d62c444a261d-0\', usage_metadata={\'input_tokens\': 405, \'output_tokens\': 32, \'total_tokens\': 437}), HumanMessage(content=\'Remember my
```

name?\', additional_kwargs={}, response_metadata={}, id=\'daba7df6-ad75-4d6b-8057-745881cea1ca\'), AIMessage(content="Of course, I remember your name, Will. I always try to pay attention to important details that users share with me. Is there anything else you\'d like to talk about or any questions you have? I\'m here to help with a wide range of topics or tasks.", additional_kwargs={}, response_metadata={\'id\': \'msg_01E41KitY74HpENRgXx94vag\', \'model\': \'claude-3-5-sonnet-20240620\', \'stop_reason\': \'end_turn\', \'stop_sequence\': None, \'usage\': {\'input_tokens\': 444, \'output_tokens\': 58}}, id=\'run-ffeaae5c-4d2d-4ddb-bd59-5d5cbf2a5af8-0\', usage_metadata={\'input_tokens\': 444, \'output_tokens\': 58, \'total_tokens\': 502})]}, next=(), config={\'configurable\': {\'thread_id\': \'1\', \'checkpoint_ns\': \'\', \'checkpoint_id\': \'1ef7d06e-93e0-6acc-8004-f2ac846575d2\'}}, metadata={\'source\': \'loop\', \'writes\': {\'chatbot\': {\'messages\': [AIMessage(content="Of course, I remember your name, Will. I always try to pay attention to important details that users share with me. Is there anything else you\'d like to talk about or any questions you have? I\'m here to help with a wide range of topics or tasks.", additional_kwargs={}, response_metadata={\'id\': \'msg_01E41KitY74HpENRgXx94vag\', \'model\': \'claude-3-5-sonnet-20240620\', \'stop_reason\': \'end_turn\', \'stop_sequence\': None, \'usage\': {\'input_tokens\': 444, \'output_tokens\': 58}}, id=\'run-ffeaae5c-4d2d-4ddb-bd59-5d5cbf2a5af8-0\', usage_metadata={\'input_tokens\': 444, \'output_tokens\': 58, \'total_tokens\': 502})]}}, \'step\': 4, \'parents\': {}}, created_at=\'2024-09-27T19:30:10.820758+00:00\', parent_config={\'configurable\': {\'thread_id\': \'1\', \'checkpoint_ns\': \'\', \'checkpoint_id\': \'1ef7d06e-859f-6206-8003-e1bd3c264b8f\'}}, tasks=())\n\nsnapshot.next # (since the graph ended this turn, `next` is empty. If you fetch a state from within a graph invocation, next tells which node will execute next)\n\n()\n\nThe snapshot above contains the current state values, corresponding config, and the next node to process. In our case, the graph has reached an END state, so next is empty.\n\nCongratulations! Your chatbot can now maintain conversation state across sessions thanks to LangGraph\'s checkpointing system. This opens up exciting possibilities for more natural, contextual interactions. LangGraph\'s checkpointing even handles arbitrarily complex graph states, which is much more expressive and powerful than simple chat memory.\n\nIn the next part, we\'ll introduce human oversight to our bot to handle situations where it may need guidance or verification before proceeding.\n\nCheck out the code snippet below to review our graph from this section.\n\nPart 4: Human-in-the-loop¶\n\nAgents can be unreliable and may need human input to successfully accomplish tasks. Similarly, for some actions, you may want to require human approval before running to ensure that everything is running as

```
intended.\n\nLangGraph\'s persistence layer supports human-in-the-loop workflows, allowing execution to pause and resume based on user feedback. The primary interface to this functionality is the interrupt function. Calling interrupt inside a node will pause execution. Execution can be resumed, together with new input from a human, by passing in a Command. interrupt is ergonomically similar to Python\'s built-in input(), with some caveats. We demonstrate an example below.\n\nFirst, start with our existing code from Part 3. We will make one change, which is to add a simple human_assistance tool accessible to the chatbot. This tool uses interrupt to receive information from a human.\n\nfrom typing import Annotated\n\nfrom langchain_anthropic import ChatAnthropic\nfrom langchain_community.tools.tavily_search import TavilySearchResults\nfrom langchain_core.tools import tool\nfrom typing_extensions import TypedDict\n\nfrom langgraph.checkpoint.memory import MemorySaver\nfrom langgraph.graph import StateGraph, START, END\nfrom langgraph.graph.message import add_messages\nfrom langgraph.prebuilt import ToolNode, tools_condition\n\nfrom langgraph.types import Command, interrupt\n\nclass State(TypedDict):\n    messages: Annotated[list, add_messages]\n\ngraph_builder = StateGraph(State)\n\n@tool\ndef human_assistance(query: str) -> str:\n    """Request assistance from a human."""\n    human_response = interrupt({"query": query})\n    return human_response["data"]\n\ntool = TavilySearchResults(max_results=2)\ntools = [tool, human_assistance]\nllm = ChatAnthropic(model="claude-3-5-sonnet-20240620")\nllm_with_tools = llm.bind_tools(tools)\n\ndef chatbot(state: State):\n    message = llm_with_tools.invoke(state["messages"])\n    # Because we will be interrupting during tool execution,\n    # we disable parallel tool calling to avoid repeating any\n    # tool invocations when we resume.\n    assert len(message.tool_calls) <= 1\n    return {"messages": [message]}\n\ngraph_builder.add_node("chatbot", chatbot)\n\ntool_node = ToolNode(tools=tools)\ngraph_builder.add_node("tools", tool_node)\ngraph_builder.add_conditional_edges(\n    "chatbot",\n    tools_condition,\n)\ngraph_builder.add_edge("tools", "chatbot")\ngraph_builder.add_edge(START, "chatbot")\n\nAPI Reference: ChatAnthropic | TavilySearchResults | tool | MemorySaver | StateGraph | START | END | add_messages | ToolNode | tools_condition | Command | interrupt\n\nTip\n\nCheck out the Human-in-the-loop section of the How-to Guides for more examples of Human-in-the-loop workflows, including how to review and edit tool calls before they are executed.\n\nWe compile the graph with a checkpointer, as before:\n\nmemory = MemorySaver()\n\ngraph = graph_builder.compile(checkpointer=memory)\n\nVisualizing the graph, we recover the same layout as before. We have just added a tool!\n\nfrom IPython.display import Image, display\n\ntry:\n    display(Image(graph.get_graph().draw_mermaid_png()))\nexcept Exception:\n    # This requires some extra dependencies and is optional\n    pass\n\n\nLet\'s now prompt the chatbot with a question that will engage the new human_assistance tool:\n\nuser_input = "I need some expert guidance for building an AI agent. Could you
```

```
request assistance for me?"\nconfig = {"configurable": {"thread_id":
"1"}}\n\nevents = graph.stream(\n    {"messages": [{"role": "user",
"content": user_input}]},\n    config,\n    stream_mode="values",\n)\
nfor event in events:\n    if "messages" in event:\n
event["messages"][-1].pretty_print()\n\
n================================[1m Human Message
[0m=================================\n\nI need some expert guidance
for building an AI agent. Could you request assistance for me?\
n==================================[1m Ai Message
[0m==================================\n\n[{\'text\': "Certainly! I\'d
be happy to request expert assistance for you regarding building an AI
agent. To do this, I\'ll use the human_assistance function to relay
your request. Let me do that for you now.", \'type\': \'text\'},
{\'id\': \'toolu_01ABUqneqnuHNuo1vhfDFQCW\', \'input\':
{\'query\': \'A user is requesting expert guidance for building an AI
agent. Could you please provide some expert advice or resources on
this
topic?\'}, \'name\': \'human_assistance\', \'type\': \'tool_use\'}]\
nTool Calls:\n  human_assistance (toolu_01ABUqneqnuHNuo1vhfDFQCW)\n
Call ID: toolu_01ABUqneqnuHNuo1vhfDFQCW\n  Args:\n    query: A user is
requesting expert guidance for building an AI agent. Could you please
provide some expert advice or resources on this topic?\n\nThe chatbot
generated a tool call, but then execution has been interrupted! Note
that if we inspect the graph state, we see that it stopped at the
tools node:\n\nsnapshot = graph.get_state(config)\nsnapshot.next\n\
n(\'tools\',)\n\nLet\'s take a closer look at the human_assistance
tool:\n\n@tool\ndef human_assistance(query: str) -> str:\n
"""Request assistance from a human."""\n    human_response =
interrupt({"query": query})\n    return human_response["data"]\n\
nSimilar to Python\'s built-in input() function, calling interrupt
inside the tool will pause execution. Progress is persisted based on
our choice of checkpointer-- so if we are persisting with Postgres, we
can resume at any time as long as the database is alive. Here we are
persisting with the in-memory checkpointer, so we can resume any time
as long as our Python kernel is running.\n\nTo resume execution, we
pass a Command object containing data expected by the tool. The format
of this data can be customized based on our needs. Here, we just need
a dict with a key "data":\n\nhuman_response = (\n    "We, the experts
are here to help! We\'d recommend you check out LangGraph to build
your agent."\n    " It\'s much more reliable and extensible than
simple autonomous agents."\n)\n\nhuman_command =
Command(resume={"data": human_response})\n\nevents =
graph.stream(human_command, config, stream_mode="values")\nfor event
in events:\n    if "messages" in event:\n        event["messages"][-
1].pretty_print()\n\n==================================[1m Ai Message
[0m==================================\n\n[{\'text\': "Certainly! I\'d
be happy to request expert assistance for you regarding building an AI
agent. To do this, I\'ll use the human_assistance function to relay
your request. Let me do that for you now.", \'type\': \'text\'},
```

{\'id\': \'toolu_01ABUqneqnuHNuo1vhfDFQCW\', \'input\': {\'query\': \'A user is requesting expert guidance for building an AI agent. Could you please provide some expert advice or resources on this topic?\'}, \'name\': \'human_assistance\', \'type\': \'tool_use\'}]\nTool Calls:\n human_assistance (toolu_01ABUqneqnuHNuo1vhfDFQCW)\n Call ID: toolu_01ABUqneqnuHNuo1vhfDFQCW\n Args:\n   query: A user is requesting expert guidance for building an AI agent. Could you please provide some expert advice or resources on this topic?\n=================================[1m Tool Message [0m=================================\nName: human_assistance\n\nWe, the experts are here to help! We\'d recommend you check out LangGraph to build your agent. It\'s much more reliable and extensible than simple autonomous agents.\n==================================[1m Ai Message [0m==================================\n\nThank you for your patience. I\'ve received some expert advice regarding your request for guidance on building an AI agent. Here\'s what the experts have suggested:\n\nThe experts recommend that you look into LangGraph for building your AI agent. They mention that LangGraph is a more reliable and extensible option compared to simple autonomous agents.\n\nLangGraph is likely a framework or library designed specifically for creating AI agents with advanced capabilities. Here are a few points to consider based on this recommendation:\n\n1. Reliability: The experts emphasize that LangGraph is more reliable than simpler autonomous agent approaches. This could mean it has better stability, error handling, or consistent performance.\n\n2. Extensibility: LangGraph is described as more extensible, which suggests that it probably offers a flexible architecture that allows you to easily add new features or modify existing ones as your agent\'s requirements evolve.\n\n3. Advanced capabilities: Given that it\'s recommended over "simple autonomous agents," LangGraph likely provides more sophisticated tools and techniques for building complex AI agents.\n\nTo get started with LangGraph, you might want to:\n\n1. Search for the official LangGraph documentation or website to learn more about its features and how to use it.\n2. Look for tutorials or guides specifically focused on building AI agents with LangGraph.\n3. Check if there are any community forums or discussion groups where you can ask questions and get support from other developers using LangGraph.\n\nIf you\'d like more specific information about LangGraph or have any questions about this recommendation, please feel free to ask, and I can request further assistance from the experts.\n\nOur input has been received and processed as a tool message. Review this call\'s LangSmith trace to see the exact work that was done in the above call. Notice that the state is loaded in the first step so that our chatbot can continue where it left off.\n\nCongrats! You\'ve used an interrupt to add human-in-the-loop execution to your chatbot, allowing for human oversight and intervention when needed. This opens up the potential UIs you can create with your AI systems. Since we have already added a checkpointer, as long as the underlying persistence layer is running,

```
the graph can be paused indefinitely and resumed at any time as if nothing had happened.\n\nHuman-in-the-loop workflows enable a variety of new workflows and user experiences. Check out this section of the How-to Guides for more examples of Human-in-the-loop workflows, including how to review and edit tool calls before they are executed.\n\nPart 5: Customizing State¶\n\nSo far, we\'ve relied on a simple state with one entry-- a list of messages. You can go far with this simple state, but if you want to define complex behavior without relying on the message list, you can add additional fields to the state. Here we will demonstrate a new scenario, in which the chatbot is using its search tool to find specific information, and forwarding them to a human for review. Let\'s have the chatbot research the birthday of an entity. We will add name and birthday keys to the state:\n\nfrom typing import Annotated\n\nfrom typing_extensions import TypedDict\n\nfrom langgraph.graph.message import add_messages\n\n\nclass State(TypedDict):\n    messages: Annotated[list, add_messages]\n    name: str\n    birthday: str\n\nAPI Reference: add_messages\n\nAdding this information to the state makes it easily accessible by other graph nodes (e.g., a downstream node that stores or processes the information), as well as the graph\'s persistence layer.\n\nHere, we will populate the state keys inside of our human_assistance tool. This allows a human to review the information before it is stored in the state. We will again use Command, this time to issue a state update from inside our tool. Read more about use cases for Command here.\n\nfrom langchain_core.messages import ToolMessage\nfrom langchain_core.tools import InjectedToolCallId, tool\n\nfrom langgraph.types import Command, interrupt\n\n\n@tool\n# Note that because we are generating a ToolMessage for a state update, we\n# generally require the ID of the corresponding tool call. We can use\n# LangChain\'s InjectedToolCallId to signal that this argument should not\n# be revealed to the model in the tool\'s schema.\ndef human_assistance(\n    name: str, birthday: str, tool_call_id: Annotated[str, InjectedToolCallId]\n) -> str:\n    """Request assistance from a human."""\n    human_response = interrupt(\n        {\n            "question": "Is this correct?",\n            "name": name,\n            "birthday": birthday,\n        },\n    )\n    # If the information is correct, update the state as-is.\n    if human_response.get("correct", "").lower().startswith("y"):\n        verified_name = name\n        verified_birthday = birthday\n        response = "Correct"\n    # Otherwise, receive information from the human reviewer.\n    else:\n        verified_name = human_response.get("name", name)\n        verified_birthday = human_response.get("birthday", birthday)\n        response = f"Made a correction: {human_response}"\n\n    # This time we explicitly update the state with a ToolMessage inside\n    # the tool.\n    state_update = {\n        "name": verified_name,\n        "birthday": verified_birthday,\n        "messages": [ToolMessage(response, tool_call_id=tool_call_id)],\n    }\n    # We return a Command object in the tool to update our state.\n    return
```

```
Command(update=state_update)\n\nAPI Reference: ToolMessage | InjectedToolCallId | tool | Command | interrupt\n\nOtherwise, the rest of our graph is the same:\n\nfrom langchain_anthropic import ChatAnthropic\nfrom langchain_community.tools.tavily_search import TavilySearchResults\n\nfrom langgraph.checkpoint.memory import MemorySaver\nfrom langgraph.graph import StateGraph, START, END\nfrom langgraph.prebuilt import ToolNode, tools_condition\n\n\ntool = TavilySearchResults(max_results=2)\ntools = [tool, human_assistance]\nllm = ChatAnthropic(model="claude-3-5-sonnet-20240620")\nllm_with_tools = llm.bind_tools(tools)\n\n\ndef chatbot(state: State):\n    message = llm_with_tools.invoke(state["messages"])\n    assert len(message.tool_calls) <= 1\n    return {"messages": [message]}\n\n\ngraph_builder = StateGraph(State)\ngraph_builder.add_node("chatbot", chatbot)\n\ntool_node = ToolNode(tools=tools)\ngraph_builder.add_node("tools", tool_node)\n\ngraph_builder.add_conditional_edges(\n    "chatbot",\n    tools_condition,\n)\ngraph_builder.add_edge("tools", "chatbot")\ngraph_builder.add_edge(START, "chatbot")\n\nmemory = MemorySaver()\ngraph = graph_builder.compile(checkpointer=memory)\n\nAPI Reference: ChatAnthropic | TavilySearchResults | MemorySaver | StateGraph | START | END | ToolNode | tools_condition\n\nLet\'s prompt our application to look up the "birthday" of the LangGraph library. We will direct the chatbot to reach out to the human_assistance tool once it has the required information. Note that setting name and birthday in the arguments for the tool, we force the chatbot to generate proposals for these fields.\n\nuser_input = (\n    "Can you look up when LangGraph was released? "\n    "When you have the answer, use the human_assistance tool for review."\n)\nconfig = {"configurable": {"thread_id": "1"}}\n\nevents = graph.stream(\n    {"messages": [{"role": "user", "content": user_input}]},\n    config,\n    stream_mode="values",\n)\nfor event in events:\n    if "messages" in event:\n        event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nCan you look up when LangGraph was released? When you have the answer, use the human_assistance tool for review.\n==================================[1m Ai Message [0m==================================\n\n[{\'text\': "Certainly! I\'ll start by searching for information about LangGraph\'s release date using the Tavily search function. Then, I\'ll use the human_assistance tool for review.", \'type\': \'text\'}, {\'id\': \'toolu_01JoXQPgTVJXiuma8xMVwqAi\', \'input\': {\'query\': \'LangGraph release date\'}, \'name\': \'tavily_search_results_json\', \'type\': \'tool_use\'}]\nTool Calls:\n  tavily_search_results_json (toolu_01JoXQPgTVJXiuma8xMVwqAi)\n Call ID: toolu_01JoXQPgTVJXiuma8xMVwqAi\n  Args:\n    query: LangGraph release date\n=================================[1m Tool Message [0m=================================\nName: tavily_search_results_json\n\n[{"url":
```

"https://blog.langchain.dev/langgraph-cloud/", "content": "We also have a new stable release of LangGraph. By LangChain 6 min read Jun 27, 2024 (Oct \'24) Edit: Since the launch of LangGraph Cloud, we now have multiple deployment options alongside LangGraph Studio - which now fall under LangGraph Platform. LangGraph Cloud is synonymous with our Cloud SaaS deployment option."}, {"url": "https://changelog.langchain.com/announcements/langgraph-cloud-deploy-at-scale-monitor-carefully-iterate-boldly", "content": "LangChain - Changelog | ☁ LangGraph Cloud: Deploy at scale, monitor LangChain LangSmith LangGraph LangChain LangSmith LangGraph LangChain LangSmith LangGraph LangChain Changelog Sign up for our newsletter to stay up to date DATE: The LangChain Team LangGraph LangGraph Cloud ☁ LangGraph Cloud: Deploy at scale, monitor carefully, iterate boldly DATE: June 27, 2024 AUTHOR: The LangChain Team LangGraph Cloud is now in closed beta, offering scalable, fault-tolerant deployment for LangGraph agents. LangGraph Cloud also includes a new playground-like studio for debugging agent failure modes and quick iteration: Join the waitlist today for LangGraph Cloud. And to learn more, read our blog post announcement or check out our docs. Subscribe By clicking subscribe, you accept our privacy policy and terms and conditions."}]\n==========[1m Ai Message [0m==========\n\n[{\'text\': "Based on the search results, it appears that LangGraph was already in existence before June 27, 2024, when LangGraph Cloud was announced. However, the search results don\'t provide a specific release date for the original LangGraph. \\n\\nGiven this information, I\'ll use the human_assistance tool to review and potentially provide more accurate information about LangGraph\'s initial release date.", \'type\': \'text\'}, {\'id\': \'toolu_01JDQAV7nPqMkHHhNs3j3XoN\', \'input\': {\'name\': \'Assistant\', \'birthday\': \'2023-01-01\'}, \'name\': \'human_assistance\', \'type\': \'tool_use\'}]\nTool Calls:\n human_assistance (toolu_01JDQAV7nPqMkHHhNs3j3XoN)\n Call ID: toolu_01JDQAV7nPqMkHHhNs3j3XoN\n Args:\n   name: Assistant\n   birthday: 2023-01-01\n\nWe\'ve hit the interrupt in the human_assistance tool again. In this case, the chatbot failed to identify the correct date, so we can supply it:\n\nhuman_command = Command(\n   resume={\n   "name": "LangGraph",\n   "birthday": "Jan 17, 2024",\n   },\n)\n\nevents = graph.stream(human_command, config, stream_mode="values")\nfor event in events:\n   if "messages" in event:\n   event["messages"][-1].pretty_print()\n\n==========[1m Ai Message [0m==========\n\n[{\'text\': "Based on the search results, it appears that LangGraph was already in existence before June 27, 2024, when LangGraph Cloud was announced. However, the search results don\'t provide a specific release date for the original LangGraph. \\n\\nGiven this information, I\'ll use the human_assistance tool to review and potentially provide more accurate information about LangGraph\'s initial release

date.", \'type\': \'text\'},
{\'id\': \'toolu_01JDQAV7nPqMkHHhNs3j3XoN\', \'input\':
{\'name\': \'Assistant\', \'birthday\': \'2023-01-
01\'}, \'name\': \'human_assistance\', \'type\': \'tool_use\'}]\nTool
Calls:\n human_assistance (toolu_01JDQAV7nPqMkHHhNs3j3XoN)\n Call ID:
toolu_01JDQAV7nPqMkHHhNs3j3XoN\n Args:\n   name: Assistant\n
birthday: 2023-01-01\n==========[1m Tool
Message [0m==========\nName: human_assistance\
n\nMade a correction: {\'name\': \'LangGraph\', \'birthday\': \'Jan
17, 2024\'}\n==========[1m Ai Message
[0m==========\n\nThank you for the human
assistance. I can now provide you with the correct information about
LangGraph\'s release date.\n\nLangGraph was initially released on
January 17, 2024. This information comes from the human assistance
correction, which is more accurate than the search results I initially
found.\n\nTo summarize:\n1. LangGraph\'s original release date:
January 17, 2024\n2. LangGraph Cloud announcement: June 27, 2024\n\
nIt\'s worth noting that LangGraph had been in development and use for
some time before the LangGraph Cloud announcement, but the official
initial release of LangGraph itself was on January 17, 2024.\n\nNote
that these fields are now reflected in the state:\n\nsnapshot =
graph.get_state(config)\n\n{k: v for k, v in snapshot.values.items()
if k in ("name", "birthday")}\n\
n{\'name\': \'LangGraph\', \'birthday\': \'Jan 17, 2024\'}\n\nThis
makes them easily accessible to downstream nodes (e.g., a node that
further processes or stores the information).\n\nManually updating
state¶\n\nLangGraph gives a high degree of control over the
application state. For instance, at any point (including when
interrupted), we can manually override a key using
graph.update_state:\n\ngraph.update_state(config, {"name": "LangGraph
(library)"})\n\n{\'configurable\': {\'thread_id\': \'1\',\
n \'checkpoint_ns\': \'\',\n \'checkpoint_id\': \'1efd4ec5-cf69-
6352-8006-9278f1730162\'}}\n\nIf we call graph.get_state, we can see
the new value is reflected:\n\nsnapshot = graph.get_state(config)\n\
n{k: v for k, v in snapshot.values.items() if k in ("name",
"birthday")}\n\n{\'name\': \'LangGraph
(library)\', \'birthday\': \'Jan 17, 2024\'}\n\nManual state updates
will even generate a trace in LangSmith. If desired, they can also be
used to control human-in-the-loop workflows, as described in this
guide. Use of the interrupt function is generally recommended instead,
as it allows data to be transmitted in a human-in-the-loop interaction
independently of state updates.\n\nCongratulations! You\'ve added
custom keys to the state to facilitate a more complex workflow, and
learned how to generate state updates from inside tools.\n\nWe\'re
almost done with the tutorial, but there is one more concept we\'d
like to review before finishing that connects checkpointing and state
updates.\n\nThis section\'s code is reproduced below for your
reference.\n\nPart 6: Time Travel¶\n\nIn a typical chat bot workflow,
the user interacts with the bot 1 or more times to accomplish a task.

```
In the previous sections, we saw how to add memory and a human-in-the-loop to be able to checkpoint our graph state and control future responses.\n\nBut what if you want to let your user start from a previous response and "branch off" to explore a separate outcome? Or what if you want users to be able to "rewind" your assistant\'s work to fix some mistakes or try a different strategy (common in applications like autonomous software engineers)?\n\nYou can create both of these experiences and more using LangGraph\'s built-in "time travel" functionality.\n\nIn this section, you will "rewind" your graph by fetching a checkpoint using the graph\'s get_state_history method. You can then resume execution at this previous point in time.\n\nFor this, let\'s use the simple chatbot with tools from Part 3:\n\nfrom typing import Annotated\n\nfrom langchain_anthropic import ChatAnthropic\nfrom langchain_community.tools.tavily_search import TavilySearchResults\nfrom langchain_core.messages import BaseMessage\nfrom typing_extensions import TypedDict\n\nfrom langgraph.checkpoint.memory import MemorySaver\nfrom langgraph.graph import StateGraph, START, END\nfrom langgraph.graph.message import add_messages\nfrom langgraph.prebuilt import ToolNode, tools_condition\n\n\nclass State(TypedDict):\n    messages: Annotated[list, add_messages]\n\n\ngraph_builder = StateGraph(State)\n\n\ntool = TavilySearchResults(max_results=2)\ntools = [tool]\nllm = ChatAnthropic(model="claude-3-5-sonnet-20240620")\nllm_with_tools = llm.bind_tools(tools)\n\n\ndef chatbot(state: State):\n    return {"messages": [llm_with_tools.invoke(state["messages"])]}\n\n\ngraph_builder.add_node("chatbot", chatbot)\n\ntool_node = ToolNode(tools=[tool])\ngraph_builder.add_node("tools", tool_node)\n\ngraph_builder.add_conditional_edges(\n    "chatbot",\n    tools_condition,\n)\ngraph_builder.add_edge("tools", "chatbot")\ngraph_builder.add_edge(START, "chatbot")\n\nmemory = MemorySaver()\ngraph = graph_builder.compile(checkpointer=memory)\n\nAPI Reference: ChatAnthropic | TavilySearchResults | BaseMessage | MemorySaver | StateGraph | START | END | add_messages | ToolNode | tools_condition\n\nLet\'s have our graph take a couple steps. Every step will be checkpointed in its state history:\n\nconfig = {"configurable": {"thread_id": "1"}}\nevents = graph.stream(\n    {\n        "messages": [\n            {\n                "role": "user",\n                "content": (\n                    "I\'m learning LangGraph. "\n                    "Could you do some research on it for me?"\n                ),\n            },\n        ],\n    },\n    config,\n    stream_mode="values",\n)\nfor event in events:\n    if "messages" in event:\n        event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nI\'m learning LangGraph. Could you do some research on it for me?\n==================================[1m Ai Message [0m==================================\n\n[{\'text\': "Certainly! I\'d be happy to research LangGraph for you. To get the most up-to-date and accurate information, I\'ll use the Tavily search engine to look this
```

up. Let me do that for you now.", \'type\': \'text\'},
{\'id\': \'toolu_01BscbfJJB9EWJFqGrN6E54e\', \'input\':
{\'query\': \'LangGraph latest information and
features\'}, \'name\': \'tavily_search_results_json\', \'type\': \'too
l_use\'}]\nTool Calls:\n tavily_search_results_json
(toolu_01BscbfJJB9EWJFqGrN6E54e)\n Call ID:
toolu_01BscbfJJB9EWJFqGrN6E54e\n Args:\n   query: LangGraph latest
information and features\n==========[1m Tool
Message [0m==========\nName:
tavily_search_results_json\n\n[{"url":
"https://blockchain.news/news/langchain-new-features-upcoming-events-
update", "content": "LangChain, a leading platform in the AI
development space, has released its latest updates, showcasing new use
cases and enhancements across its ecosystem. According to the
LangChain Blog, the updates cover advancements in LangGraph Cloud,
LangSmith\'s self-improving evaluators, and revamped documentation for
LangGraph."}, {"url": "https://blog.langchain.dev/langgraph-platform-
announce/", "content": "With these learnings under our belt, we
decided to couple some of our latest offerings under LangGraph
Platform. LangGraph Platform today includes LangGraph Server,
LangGraph Studio, plus the CLI and SDK. ... we added features in
LangGraph Server to deliver on a few key value areas. Below, we\'ll
focus on these aspects of LangGraph Platform."}]\
n==========[1m Ai Message
[0m==========\n\nThank you for your patience.
I\'ve found some recent information about LangGraph for you. Let me
summarize the key points:\n\n1. LangGraph is part of the LangChain
ecosystem, which is a leading platform in AI development.\n\n2. Recent
updates and features of LangGraph include:\n\n   a. LangGraph Cloud:
This seems to be a cloud-based version of LangGraph, though specific
details weren\'t provided in the search results.\n\n   b. LangGraph
Platform: This is a newly introduced concept that combines several
offerings:\n   - LangGraph Server\n   - LangGraph Studio\n
- CLI (Command Line Interface)\n   - SDK (Software Development
Kit)\n\n3. LangGraph Server: This component has received new features
to enhance its value proposition, though the specific features
weren\'t detailed in the search results.\n\n4. LangGraph Studio: This
appears to be a new tool in the LangGraph ecosystem, likely providing
a graphical interface for working with LangGraph.\n\n5. Documentation:
The LangGraph documentation has been revamped, which should make it
easier for learners like yourself to understand and use the tool.\n\
n6. Integration with LangSmith: While not directly part of LangGraph,
LangSmith (another tool in the LangChain ecosystem) now features self-
improving evaluators, which might be relevant if you\'re using
LangGraph as part of a larger LangChain project.\n\nAs you\'re
learning LangGraph, it would be beneficial to:\n\n1. Check out the
official LangChain documentation, especially the newly revamped
LangGraph sections.\n2. Explore the different components of the
LangGraph Platform (Server, Studio, CLI, and SDK) to see which best

fits your learning needs.\n3. Keep an eye on LangGraph Cloud developments, as cloud-based solutions often provide an easier starting point for learners.\n4. Consider how LangGraph fits into the broader LangChain ecosystem, especially its interaction with tools like LangSmith.\n\nIs there any specific aspect of LangGraph you\'d like to know more about? I\'d be happy to do a more focused search on particular features or use cases.\n\nevents = graph.stream(\n   {\n "messages": [\n   {\n   "role": "user",\n "content": (\n   "Ya that\'s helpful. Maybe I\'ll "\n "build an autonomous agent with it!"\n   ),\
n   },\n   ],\n   },\n   config,\n stream_mode="values",\n)\nfor event in events:\n   if "messages" in event:\n   event["messages"][-1].pretty_print()\n\
n================================[1m Human Message [0m=================================\n\nYa that\'s helpful. Maybe I\'ll build an autonomous agent with it!\
n==================================[1m Ai Message [0m==================================\n\n[{\'text\': "That\'s an exciting idea! Building an autonomous agent with LangGraph is indeed a great application of this technology. LangGraph is particularly well-suited for creating complex, multi-step AI workflows, which is perfect for autonomous agents. Let me gather some more specific information about using LangGraph for building autonomous agents.", \'type\': \'text\'},
{\'id\': \'toolu_01QWNHhUaeeWcGXvA4eHT7Zo\', \'input\': {\'query\': \'Building autonomous agents with LangGraph examples and tutorials\'}, \'name\': \'tavily_search_results_json\', \'type\': \'tool_use\'}]\nTool Calls:\n tavily_search_results_json (toolu_01QWNHhUaeeWcGXvA4eHT7Zo)\n Call ID: toolu_01QWNHhUaeeWcGXvA4eHT7Zo\n Args:\n   query: Building autonomous agents with LangGraph examples and tutorials\
n=================================[1m Tool Message [0m=================================\nName: tavily_search_results_json\n\n[{"url": "https://towardsdatascience.com/building-autonomous-multi-tool-agents-with-gemini-2-0-and-langgraph-ad3d7bd5e79d", "content": "Building Autonomous Multi-Tool Agents with Gemini 2.0 and LangGraph | by Youness Mansar | Jan, 2025 | Towards Data Science Building Autonomous Multi-Tool Agents with Gemini 2.0 and LangGraph A practical tutorial with full code examples for building and running multi-tool agents Towards Data Science LLMs are remarkable — they can memorize vast amounts of information, answer general knowledge questions, write code, generate stories, and even fix your grammar. In this tutorial, we are going to build a simple LLM agent that is equipped with four tools that it can use to answer a user’s question. This Agent will have the following specifications: Follow Published in Towards Data Science ---------- Your home for data science and AI. Follow Follow Follow"}, {"url": "https://github.com/anmolaman20/Tools_and_Agents", "content": "GitHub

- anmolaman20/Tools_and_Agents: This repository provides resources for building AI agents using Langchain and Langgraph. This repository provides resources for building AI agents using Langchain and Langgraph. This repository provides resources for building AI agents using Langchain and Langgraph. This repository serves as a comprehensive guide for building AI-powered agents using Langchain and Langgraph. It provides hands-on examples, practical tutorials, and resources for developers and AI enthusiasts to master building intelligent systems and workflows. AI Agent Development: Gain insights into creating intelligent systems that think, reason, and adapt in real time. This repository is ideal for AI practitioners, developers exploring language models, or anyone interested in building intelligent systems. This repository provides resources for building AI agents using Langchain and Langgraph."}]\
n==========[1m Ai Message [0m==========\n\nGreat idea! Building an autonomous agent with LangGraph is definitely an exciting project. Based on the latest information I\'ve found, here are some insights and tips for building autonomous agents with LangGraph:\n\n1. Multi-Tool Agents: LangGraph is particularly well-suited for creating autonomous agents that can use multiple tools. This allows your agent to have a diverse set of capabilities and choose the right tool for each task.\n\n2. Integration with Large Language Models (LLMs): You can combine LangGraph with powerful LLMs like Gemini 2.0 to create more intelligent and capable agents. The LLM can serve as the "brain" of your agent, making decisions and generating responses.\n\n3. Workflow Management: LangGraph excels at managing complex, multi-step AI workflows. This is crucial for autonomous agents that need to break down tasks into smaller steps and execute them in the right order.\n\n4. Practical Tutorials Available: There are tutorials available that provide full code examples for building and running multi-tool agents. These can be incredibly helpful as you start your project.\n\n5. Langchain Integration: LangGraph is often used in conjunction with Langchain. This combination provides a powerful framework for building AI agents, offering features like memory management, tool integration, and prompt management.\n\n6. GitHub Resources: There are repositories available (like the one by anmolaman20) that provide comprehensive resources for building AI agents using Langchain and LangGraph. These can be valuable references as you develop your agent.\n\n7. Real-time Adaptation: LangGraph allows you to create agents that can think, reason, and adapt in real-time, which is crucial for truly autonomous behavior.\n\n8. Customization: You can equip your agent with specific tools tailored to your use case. For example, you might include tools for web searching, data analysis, or interacting with specific APIs.\n\nTo get started with your autonomous agent project:\n\n1. Familiarize yourself with LangGraph\'s documentation and basic concepts.\n2. Look into tutorials that specifically deal with building autonomous agents, like the one mentioned from Towards Data Science.\n3. Decide on the specific capabilities you want your agent to have

and identify the tools it will need.\n4. Start with a simple agent and gradually add complexity as you become more comfortable with the framework.\n5. Experiment with different LLMs to find the one that works best for your use case.\n6. Pay attention to how you structure the agent\'s decision-making process and workflow.\n7. Don\'t forget to implement proper error handling and safety measures, especially if your agent will be interacting with external systems or making important decisions.\n\nBuilding an autonomous agent is an iterative process, so be prepared to refine and improve your agent over time. Good luck with your project! If you need any more specific information as you progress, feel free to ask.\n\nNow that we\'ve had the agent take a couple steps, we can replay the full state history to see everything that occurred.\n\nto_replay = None\nfor state in graph.get_state_history(config):\n   print("Num Messages: ", len(state.values["messages"]), "Next: ", state.next)\n   print("-" * 80)\n   if len(state.values["messages"]) == 6:\n   # We are somewhat arbitrarily selecting a specific state based on the number of chat messages in the state.\n   to_replay = state\n\nNum Messages: 8 Next: ()\
n----------\nNum Messages: 7 Next: (\'chatbot\',)\
n----------\nNum Messages: 6 Next: (\'tools\',)\
n----------\nNum Messages: 5 Next: (\'chatbot\',)\
n----------\nNum Messages: 4 Next: (\'__start__\',)\
n----------\nNum Messages: 4 Next: ()\
n----------\nNum Messages: 3 Next: (\'chatbot\',)\
n----------\nNum Messages: 2 Next: (\'tools\',)\
n----------\nNum Messages: 1 Next: (\'chatbot\',)\
n----------\nNum Messages: 0 Next: (\'__start__\',)\
n----------\n\nNotice that checkpoints are saved for every step of the graph. This spans invocations so you can rewind across a full thread\'s history. We\'ve picked out to_replay as a state to resume from. This is the state after the chatbot node in the second graph invocation above.\n\nResuming from this point should call the action node next.\n\nprint(to_replay.next)\nprint(to_replay.config)\n\
n(\'tools\',)\n{\'configurable\':
{\'thread_id\': \'1\', \'checkpoint_ns\': \'\', \'checkpoint_id\': \'1efd43e3-0c1f-6c4e-8006-891877d65740\'}}\n\nNotice that the checkpoint\'s config (to_replay.config) contains a checkpoint_id timestamp. Providing this checkpoint_id value tells LangGraph\'s checkpointer to load the state from that moment in time. Let\'s try it

below:\n\n# The `checkpoint_id` in the `to_replay.config` corresponds to a state we\'ve persisted to our checkpointer.\nfor event in graph.stream(None, to_replay.config, stream_mode="values"):\n   if "messages" in event:\n   event["messages"][-1].pretty_print()\n\n==========[1m Ai Message [0m==========\n\n[{\'text\': "That\'s an exciting idea! Building an autonomous agent with LangGraph is indeed a great application of this technology. LangGraph is particularly well-suited for creating complex, multi-step AI workflows, which is perfect for autonomous agents. Let me gather some more specific information about using LangGraph for building autonomous agents.", \'type\': \'text\'}, {\'id\': \'toolu_01QWNHhUaeeWcGXvA4eHT7Zo\', \'input\': {\'query\': \'Building autonomous agents with LangGraph examples and tutorials\'}, \'name\': \'tavily_search_results_json\', \'type\': \'tool_use\'}]\nTool Calls:\n tavily_search_results_json (toolu_01QWNHhUaeeWcGXvA4eHT7Zo)\n Call ID: toolu_01QWNHhUaeeWcGXvA4eHT7Zo\n Args:\n   query: Building autonomous agents with LangGraph examples and tutorials\n==========[1m Tool Message [0m==========\nName: tavily_search_results_json\n\n[{"url": "https://towardsdatascience.com/building-autonomous-multi-tool-agents-with-gemini-2-0-and-langgraph-ad3d7bd5e79d", "content": "Building Autonomous Multi-Tool Agents with Gemini 2.0 and LangGraph | by Youness Mansar | Jan, 2025 | Towards Data Science Building Autonomous Multi-Tool Agents with Gemini 2.0 and LangGraph A practical tutorial with full code examples for building and running multi-tool agents Towards Data Science LLMs are remarkable — they can memorize vast amounts of information, answer general knowledge questions, write code, generate stories, and even fix your grammar. In this tutorial, we are going to build a simple LLM agent that is equipped with four tools that it can use to answer a user’s question. This Agent will have the following specifications: Follow Published in Towards Data Science ---------- Your home for data science and AI. Follow Follow Follow"}, {"url": "https://github.com/anmolaman20/Tools_and_Agents", "content": "GitHub - anmolaman20/Tools_and_Agents: This repository provides resources for building AI agents using Langchain and Langgraph. This repository provides resources for building AI agents using Langchain and Langgraph. This repository provides resources for building AI agents using Langchain and Langgraph. This repository serves as a comprehensive guide for building AI-powered agents using Langchain and Langgraph. It provides hands-on examples, practical tutorials, and resources for developers and AI enthusiasts to master building intelligent systems and workflows. AI Agent Development: Gain insights into creating intelligent systems that think, reason, and adapt in real time. This repository is ideal for AI practitioners, developers exploring language models, or anyone interested in building

intelligent systems. This repository provides resources for building AI agents using Langchain and Langgraph."}]\n==========[1m Ai Message [0m==========\n\nGreat idea! Building an autonomous agent with LangGraph is indeed an excellent way to apply and deepen your understanding of the technology. Based on the search results, I can provide you with some insights and resources to help you get started:\n\n1. Multi-Tool Agents:\n   LangGraph is well-suited for building autonomous agents that can use multiple tools. This allows your agent to have a variety of capabilities and choose the appropriate tool based on the task at hand.\n\n2. Integration with Large Language Models (LLMs):\n   There\'s a tutorial that specifically mentions using Gemini 2.0 (Google\'s LLM) with LangGraph to build autonomous agents. This suggests that LangGraph can be integrated with various LLMs, giving you flexibility in choosing the language model that best fits your needs.\n\n3. Practical Tutorials:\n There are tutorials available that provide full code examples for building and running multi-tool agents. These can be invaluable as you start your project, giving you a concrete starting point and demonstrating best practices.\n\n4. GitHub Resources:\n   There\'s a GitHub repository (github.com/anmolaman20/Tools_and_Agents) that provides resources for building AI agents using both Langchain and Langgraph. This could be a great resource for code examples, tutorials, and understanding how LangGraph fits into the broader LangChain ecosystem.\n\n5. Real-Time Adaptation:\n   The resources mention creating intelligent systems that can think, reason, and adapt in real-time. This is a key feature of advanced autonomous agents and something you can aim for in your project.\n\n6. Diverse Applications:\n   The materials suggest that these techniques can be applied to various tasks, from answering questions to potentially more complex decision-making processes.\n\nTo get started with your autonomous agent project using LangGraph, you might want to:\n\n1. Review the tutorials mentioned, especially those with full code examples.\n2. Explore the GitHub repository for hands-on examples and resources.\n3. Decide on the specific tasks or capabilities you want your agent to have.\n4. Choose an LLM to integrate with LangGraph (like GPT, Gemini, or others).\n5. Start with a simple agent that uses one or two tools, then gradually expand its capabilities.\n6. Implement decision-making logic to help your agent choose between different tools or actions.\n7. Test your agent thoroughly with various inputs and scenarios to ensure robust performance.\n\nRemember, building an autonomous agent is an iterative process. Start simple and gradually increase complexity as you become more comfortable with LangGraph and its capabilities.\n\nWould you like more information on any specific aspect of building your autonomous agent with LangGraph?\n\nNotice that the graph resumed execution from the **action** node. You can tell this is the case since the first value printed above is the response from our search engine tool.\n\nCongratulations! You\'ve now used time-travel checkpoint traversal in

LangGraph. Being able to rewind and explore alternative paths opens up a world of possibilities for debugging, experimentation, and interactive applications.\n\nNext Steps¶\n\nTake your journey further by exploring deployment and advanced features:\n\nServer Quickstart¶\n\nLangGraph Server Quickstart: Launch a LangGraph server locally and interact with it using the REST API and LangGraph Studio Web UI.\n\nLangGraph Cloud¶\n\nLangGraph Cloud QuickStart: Deploy your LangGraph app using LangGraph Cloud.\n\nLangGraph Framework¶\n\nLangGraph Concepts: Learn the foundational concepts of LangGraph.\n\nLangGraph How-to Guides: Guides for common tasks with LangGraph.\n\nLangGraph Platform¶\n\nExpand your knowledge with these resources:\n\nLangGraph Platform Concepts: Understand the foundational concepts of the LangGraph Platform.\n\nLangGraph Platform How-to Guides: Guides for common tasks with LangGraph Platform.\n\nComments')

docs

[Document(metadata={'source': 'https://langchain-ai.github.io/langgraph/tutorials/introduction/'}, page_content='□ LangGraph Quickstart¶\n\nIn this tutorial, we will build a support chatbot in LangGraph that can:\n\n□ Answer common questions by searching the web □ Maintain conversation state across calls □ Route complex queries to a human for review □ Use custom state to control its behavior □ Rewind and explore alternative conversation paths\n\nWe\'ll start with a basic chatbot and progressively add more sophisticated capabilities, introducing key LangGraph concepts along the way. Let’s dive in! □\n\nSetup¶\n\nFirst, install the required packages and configure your environment:\n\n%%capture --no-stderr\n%pip install -U langgraph langsmith langchain_anthropic\n\nimport getpass\nimport os\n\n\ndef _set_env(var: str):\n   if not os.environ.get(var):\n   os.environ[var] = getpass.getpass(f"{var}: ")\n\n\n_set_env("ANTHROPIC_API_KEY")\n\nSet up LangSmith for LangGraph development\n\nSign up for LangSmith to quickly spot issues and improve the performance of your LangGraph projects. LangSmith lets you use trace data to debug, test, and monitor your LLM apps built with LangGraph — read more about how to get started here.\n\nPart 1: Build a Basic Chatbot¶\n\nWe\'ll first create a simple chatbot using LangGraph. This chatbot will respond directly to user messages. Though simple, it will illustrate the core concepts of building with LangGraph. By the end of this section, you will have a built rudimentary chatbot.\n\nStart by creating a StateGraph. A StateGraph object defines the structure of our chatbot as a "state machine". We\'ll add nodes to represent the llm and functions our chatbot can call and edges to specify how the bot should transition between these functions.\n\nfrom typing import Annotated\n\nfrom typing_extensions import TypedDict\n\nfrom langgraph.graph import StateGraph, START, END\nfrom langgraph.graph.message import add_messages\n\n\nclass State(TypedDict):\n   # Messages have the type "list". The `add_messages` function\n   # in the annotation defines how this state key should be updated\n   # (in this case, it

```
appends messages to the list, rather than overwriting them)\n
messages: Annotated[list, add_messages]\n\n\ngraph_builder =
StateGraph(State)\n\nAPI Reference: StateGraph | START | END |
add_messages\n\nOur graph can now handle two key tasks:\n\nEach node
can receive the current State as input and output an update to the
state.\n\nUpdates to messages will be appended to the existing list
rather than overwriting it, thanks to the prebuilt add_messages
function used with the Annotated syntax.\n\nConcept\n\nWhen defining a
graph, the first step is to define its State. The State includes the
graph\'s schema and reducer functions that handle state updates. In
our example, State is a TypedDict with one key: messages. The
add_messages reducer function is used to append new messages to the
list instead of overwriting it. Keys without a reducer annotation will
overwrite previous values. Learn more about state, reducers, and
related concepts in this guide.\n\nNext, add a "chatbot" node. Nodes
represent units of work. They are typically regular python functions.\
n\nfrom langchain_anthropic import ChatAnthropic\n\nllm =
ChatAnthropic(model="claude-3-5-sonnet-20240620")\n\n\ndef
chatbot(state: State):\n    return {"messages":
[llm.invoke(state["messages"])]}\n\n\n# The first argument is the
unique node name\n# The second argument is the function or object that
will be called whenever\n# the node is used.\
ngraph_builder.add_node("chatbot", chatbot)\n\nAPI Reference:
ChatAnthropic\n\nNotice how the chatbot node function takes the
current State as input and returns a dictionary containing an updated
messages list under the key "messages". This is the basic pattern for
all LangGraph node functions.\n\nThe add_messages function in our
State will append the llm\'s response messages to whatever messages
are already in the state.\n\nNext, add an entry point. This tells our
graph where to start its work each time we run it.\n\
ngraph_builder.add_edge(START, "chatbot")\n\nSimilarly, set a finish
point. This instructs the graph "any time this node is run, you can
exit."\n\ngraph_builder.add_edge("chatbot", END)\n\nFinally, we\'ll
want to be able to run our graph. To do so, call "compile()" on the
graph builder. This creates a "CompiledGraph" we can use invoke on our
state.\n\ngraph = graph_builder.compile()\n\nYou can visualize the
graph using the get_graph method and one of the "draw" methods, like
draw_ascii or draw_png. The draw methods each require additional
dependencies.\n\nfrom IPython.display import Image, display\n\ntry:\n
display(Image(graph.get_graph().draw_mermaid_png()))\nexcept
Exception:\n    # This requires some extra dependencies and is
optional\n    pass\n\n\n\nNow let\'s run the chatbot!\n\nTip: You can
exit the chat loop at any time by typing "quit", "exit", or "q".\n\
ndef stream_graph_updates(user_input: str):\n    for event in
graph.stream({"messages": [{"role": "user", "content":
user_input}]}):\n        for value in event.values():\n
print("Assistant:", value["messages"][-1].content)\n\n\nwhile True:\n
try:\n        user_input = input("User: ")\n        if
user_input.lower() in ["quit", "exit", "q"]:\n
```

print("Goodbye!")\n   break\n
stream_graph_updates(user_input)\n   except:\n   # fallback if
input() is not available\n   user_input = "What do you know about
LangGraph?"\n   print("User: " + user_input)\n
stream_graph_updates(user_input)\n   break\n\nAssistant:
LangGraph is a library designed to help build stateful multi-agent
applications using language models. It provides tools for creating
workflows and state machines to coordinate multiple AI agents or
language model interactions. LangGraph is built on top of LangChain,
leveraging its components while adding graph-based coordination
capabilities. It\'s particularly useful for developing more complex,
stateful AI applications that go beyond simple query-response
interactions.\nGoodbye!\n\nCongratulations! You\'ve built your first
chatbot using LangGraph. This bot can engage in basic conversation by
taking user input and generating responses using an LLM. You can
inspect a LangSmith Trace for the call above at the provided link.\n\
nHowever, you may have noticed that the bot\'s knowledge is limited to
what\'s in its training data. In the next part, we\'ll add a web
search tool to expand the bot\'s knowledge and make it more capable.\
n\nBelow is the full code for this section for your reference:\n\nPart
2: Enhancing the Chatbot with Tools¶\n\nTo handle queries our chatbot
can\'t answer "from memory", we\'ll integrate a web search tool. Our
bot can use this tool to find relevant information and provide better
responses.\n\nRequirements¶\n\nBefore we start, make sure you have the
necessary packages installed and API keys set up:\n\nFirst, install
the requirements to use the Tavily Search Engine, and set your
TAVILY_API_KEY.\n\n%%capture --no-stderr\n%pip install -U tavily-
python langchain_community\n\n_set_env("TAVILY_API_KEY")\n\
nTAVILY_API_KEY: ·····\n\nNext, define the tool:\n\nfrom
langchain_community.tools.tavily_search import TavilySearchResults\n\
ntool = TavilySearchResults(max_results=2)\ntools = [tool]\
ntool.invoke("What\'s a \'node\' in LangGraph?")\n\nAPI Reference:
TavilySearchResults\n\n[{\'url\':
\'https://medium.com/@cplog/introduction-to-langgraph-a-beginners-
guide-14f9be027141\',\n \'content\': \'Nodes: Nodes are the building
blocks of your LangGraph. Each node represents a function or a
computation step. You define nodes to perform specific tasks, such as
processing input, making ...\'},\n {\'url\':
\'https://saksheepatil05.medium.com/demystifying-langgraph-a-beginner-
friendly-dive-into-langgraph-concepts-5ffe890ddac0\',\
n \'content\': \'Nodes (Tasks): Nodes are like the workstations on
the assembly line. Each node performs a specific task on the product.
In LangGraph, nodes are Python functions that take the current state,
do some work, and return an updated state. Next, we define the nodes,
each representing a task in our sandwich-making process.\'}]\n\nThe
results are page summaries our chat bot can use to answer questions.\
n\nNext, we\'ll start defining our graph. The following is all the
same as in Part 1, except we have added bind_tools on our LLM. This
lets the LLM know the correct JSON format to use if it wants to use

```
our search engine.\n\nfrom typing import Annotated\n\nfrom
langchain_anthropic import ChatAnthropic\nfrom typing_extensions
import TypedDict\n\nfrom langgraph.graph import StateGraph, START,
END\nfrom langgraph.graph.message import add_messages\n\n\nclass
State(TypedDict):\n    messages: Annotated[list, add_messages]\n\n\
ngraph_builder = StateGraph(State)\n\n\nllm =
ChatAnthropic(model="claude-3-5-sonnet-20240620")\n# Modification:
tell the LLM which tools it can call\nllm_with_tools =
llm.bind_tools(tools)\n\n\ndef chatbot(state: State):\n    return
{"messages": [llm_with_tools.invoke(state["messages"])]}\n\n\
ngraph_builder.add_node("chatbot", chatbot)\n\nAPI Reference:
ChatAnthropic | StateGraph | START | END | add_messages\n\nNext we
need to create a function to actually run the tools if they are
called. We\'ll do this by adding the tools to a new node.\n\nBelow, we
implement a BasicToolNode that checks the most recent message in the
state and calls tools if the message contains tool_calls. It relies on
the LLM\'s tool_calling support, which is available in Anthropic,
OpenAI, Google Gemini, and a number of other LLM providers.\n\nWe will
later replace this with LangGraph\'s prebuilt ToolNode to speed things
up, but building it ourselves first is instructive.\n\nimport json\n\
nfrom langchain_core.messages import ToolMessage\n\n\nclass
BasicToolNode:\n    """A node that runs the tools requested in the
last AIMessage."""\n\n    def __init__(self, tools: list) -> None:\n
self.tools_by_name = {tool.name: tool for tool in tools}\n\n    def
__call__(self, inputs: dict):\n        if messages :=
inputs.get("messages", []):\n            message = messages[-1]\n
else:\n            raise ValueError("No message found in input")\n
outputs = []\n        for tool_call in message.tool_calls:\n
tool_result = self.tools_by_name[tool_call["name"]].invoke(\n
tool_call["args"]\n            )\n            outputs.append(\n
ToolMessage(\n                    content=json.dumps(tool_result),\n
name=tool_call["name"],\n
tool_call_id=tool_call["id"],\n                )\n            )\n
return {"messages": outputs}\n\n\ntool_node =
BasicToolNode(tools=[tool])\ngraph_builder.add_node("tools",
tool_node)\n\nAPI Reference: ToolMessage\n\nWith the tool node added,
we can define the conditional_edges.\n\nRecall that edges route the
control flow from one node to the next. Conditional edges usually
contain "if" statements to route to different nodes depending on the
current graph state. These functions receive the current graph state
and return a string or list of strings indicating which node(s) to
call next.\n\nBelow, call define a router function called route_tools,
that checks for tool_calls in the chatbot\'s output. Provide this
function to the graph by calling add_conditional_edges, which tells
the graph that whenever the chatbot node completes to check this
function to see where to go next.\n\nThe condition will route to tools
if tool calls are present and END if not.\n\nLater, we will replace
this with the prebuilt tools_condition to be more concise, but
implementing it ourselves first makes things more clear.\n\ndef
```

```
route_tools(\n    state: State,\n):\n    """\n    Use in the
conditional_edge to route to the ToolNode if the last message\n    has
tool calls. Otherwise, route to the end.\n    """\n    if
isinstance(state, list):\n        ai_message = state[-1]\n    elif
messages := state.get("messages", []):\n        ai_message =
messages[-1]\n    else:\n        raise ValueError(f"No messages found
in input state to tool_edge: {state}")\n    if hasattr(ai_message,
"tool_calls") and len(ai_message.tool_calls) > 0:\n        return
"tools"\n    return END\n\n\n# The `tools_condition` function returns
"tools" if the chatbot asks to use a tool, and "END" if\n# it is fine
directly responding. This conditional routing defines the main agent
loop.\ngraph_builder.add_conditional_edges(\n    "chatbot",\n
route_tools,\n    # The following dictionary lets you tell the graph
to interpret the condition\'s outputs as a specific node\n    # It
defaults to the identity function, but if you\n    # want to use a
node named something else apart from "tools",\n    # You can update
the value of the dictionary to something else\n    # e.g., "tools":
"my_tools"\n    {"tools": "tools", END: END},\n)\n# Any time a tool is
called, we return to the chatbot to decide the next step\
ngraph_builder.add_edge("tools", "chatbot")\
ngraph_builder.add_edge(START, "chatbot")\ngraph =
graph_builder.compile()\n\nNotice that conditional edges start from a
single node. This tells the graph "any time the \'chatbot\' node runs,
either go to \'tools\' if it calls a tool, or end the loop if it
responds directly.\n\nLike the prebuilt tools_condition, our function
returns the END string if no tool calls are made. When the graph
transitions to END, it has no more tasks to complete and ceases
execution. Because the condition can return END, we don\'t need to
explicitly set a finish_point this time. Our graph already has a way
to finish!\n\nLet\'s visualize the graph we\'ve built. The following
function has some additional dependencies to run that are unimportant
for this tutorial.\n\nfrom IPython.display import Image, display\n\
ntry:\n    display(Image(graph.get_graph().draw_mermaid_png()))\
nexcept Exception:\n    # This requires some extra dependencies and is
optional\n    pass\n\n\nNow we can ask the bot questions outside its
training data.\n\nwhile True:\n    try:\n        user_input =
input("User: ")\n        if user_input.lower() in ["quit", "exit",
"q"]:\n            print("Goodbye!")\n            break\n\n
stream_graph_updates(user_input)\n    except:\n        # fallback if
input() is not available\n        user_input = "What do you know about
LangGraph?"\n        print("User: " + user_input)\n
stream_graph_updates(user_input)\n        break\n\nAssistant:
[{\'text\': "To provide you with accurate and up-to-date information
about LangGraph, I\'ll need to search for the latest details. Let me
do that for you.", \'type\': \'text\'},
{\'id\': \'toolu_01Q588CszHaSvvP2MxRq9zRD\', \'input\':
{\'query\': \'LangGraph AI tool
information\'}, \'name\': \'tavily_search_results_json\', \'type\': \'
tool_use\'}]\nAssistant: [{"url":
```

"https://www.langchain.com/langgraph", "content": "LangGraph sets the foundation for how we can build and scale AI workloads \\u2014 from conversational agents, complex task automation, to custom LLM-backed experiences that \'just work\'. The next chapter in building complex production-ready features with LLMs is agentic, and with LangGraph and LangSmith, LangChain delivers an out-of-the-box solution ..."}, {"url": "https://github.com/langchain-ai/langgraph", "content": "Overview. LangGraph is a library for building stateful, multi-actor applications with LLMs, used to create agent and multi-agent workflows. Compared to other LLM frameworks, it offers these core benefits: cycles, controllability, and persistence. LangGraph allows you to define flows that involve cycles, essential for most agentic architectures ..."}]\nAssistant: Based on the search results, I can provide you with information about LangGraph:\n\n1. Purpose:\n LangGraph is a library designed for building stateful, multi-actor applications with Large Language Models (LLMs). It\'s particularly useful for creating agent and multi-agent workflows.\n\n2. Developer:\n LangGraph is developed by LangChain, a company known for its tools and frameworks in the AI and LLM space.\n\n3. Key Features:\n - Cycles: LangGraph allows the definition of flows that involve cycles, which is essential for most agentic architectures.\n - Controllability: It offers enhanced control over the application flow.\n - Persistence: The library provides ways to maintain state and persistence in LLM-based applications.\n\n4. Use Cases:\n LangGraph can be used for various applications, including:\n - Conversational agents\n - Complex task automation\n - Custom LLM-backed experiences\n\n5. Integration:\n LangGraph works in conjunction with LangSmith, another tool by LangChain, to provide an out-of-the-box solution for building complex, production-ready features with LLMs.\n\n6. Significance:\n LangGraph is described as setting the foundation for building and scaling AI workloads. It\'s positioned as a key tool in the next chapter of LLM-based application development, particularly in the realm of agentic AI.\n\n7. Availability:\n LangGraph is open-source and available on GitHub, which suggests that developers can access and contribute to its codebase.\n\n8. Comparison to Other Frameworks:\n LangGraph is noted to offer unique benefits compared to other LLM frameworks, particularly in its ability to handle cycles, provide controllability, and maintain persistence.\n\nLangGraph appears to be a significant tool in the evolving landscape of LLM-based application development, offering developers new ways to create more complex, stateful, and interactive AI systems.\nGoodbye!\n\nCongrats! You\'ve created a conversational agent in langgraph that can use a search engine to retrieve updated information when needed. Now it can handle a wider range of user queries. To inspect all the steps your agent just took, check out this LangSmith trace.\n\nOur chatbot still can\'t remember past interactions on its own, limiting its ability to have coherent, multi-turn conversations. In the next part, we\'ll add memory to address this.\n\nThe full code for the graph we\'ve created in this

```
section is reproduced below, replacing our BasicToolNode for the
prebuilt ToolNode, and our route_tools condition with the prebuilt
tools_condition\n\nPart 3: Adding Memory to the Chatbot¶\n\nOur
chatbot can now use tools to answer user questions, but it doesn\'t
remember the context of previous interactions. This limits its ability
to have coherent, multi-turn conversations.\n\nLangGraph solves this
problem through persistent checkpointing. If you provide a
checkpointer when compiling the graph and a thread_id when calling
your graph, LangGraph automatically saves the state after each step.
When you invoke the graph again using the same thread_id, the graph
loads its saved state, allowing the chatbot to pick up where it left
off.\n\nWe will see later that checkpointing is much more powerful
than simple chat memory - it lets you save and resume complex state at
any time for error recovery, human-in-the-loop workflows, time travel
interactions, and more. But before we get too ahead of ourselves,
let\'s add checkpointing to enable multi-turn conversations.\n\nTo get
started, create a MemorySaver checkpointer.\n\nfrom
langgraph.checkpoint.memory import MemorySaver\n\nmemory =
MemorySaver()\n\nAPI Reference: MemorySaver\n\nNotice we\'re using an
in-memory checkpointer. This is convenient for our tutorial (it saves
it all in-memory). In a production application, you would likely
change this to use SqliteSaver or PostgresSaver and connect to your
own DB.\n\nNext define the graph. Now that you\'ve already built your
own BasicToolNode, we\'ll replace it with LangGraph\'s prebuilt
ToolNode and tools_condition, since these do some nice things like
parallel API execution. Apart from that, the following is all copied
from Part 2.\n\nfrom typing import Annotated\n\nfrom
langchain_anthropic import ChatAnthropic\nfrom
langchain_community.tools.tavily_search import TavilySearchResults\
nfrom langchain_core.messages import BaseMessage\nfrom
typing_extensions import TypedDict\n\nfrom langgraph.graph import
StateGraph, START, END\nfrom langgraph.graph.message import
add_messages\nfrom langgraph.prebuilt import ToolNode,
tools_condition\n\n\nclass State(TypedDict):\n    messages:
Annotated[list, add_messages]\n\n\ngraph_builder = StateGraph(State)\
n\n\ntool = TavilySearchResults(max_results=2)\ntools = [tool]\nllm =
ChatAnthropic(model="claude-3-5-sonnet-20240620")\nllm_with_tools =
llm.bind_tools(tools)\n\n\ndef chatbot(state: State):\n    return
{"messages": [llm_with_tools.invoke(state["messages"])]}\n\n\
ngraph_builder.add_node("chatbot", chatbot)\n\ntool_node =
ToolNode(tools=[tool])\ngraph_builder.add_node("tools", tool_node)\n\
ngraph_builder.add_conditional_edges(\n    "chatbot",\n
tools_condition,\n)\n# Any time a tool is called, we return to the
chatbot to decide the next step\ngraph_builder.add_edge("tools",
"chatbot")\ngraph_builder.add_edge(START, "chatbot")\n\nAPI Reference:
ChatAnthropic | TavilySearchResults | BaseMessage | StateGraph | START
| END | add_messages | ToolNode | tools_condition\n\nFinally, compile
the graph with the provided checkpointer.\n\ngraph =
graph_builder.compile(checkpointer=memory)\n\nNotice the connectivity
```

```
of the graph hasn\'t changed since Part 2. All we are doing is checkpointing the State as the graph works through each node.\n\nfrom IPython.display import Image, display\n\ntry:\n    display(Image(graph.get_graph().draw_mermaid_png()))\nexcept Exception:\n    # This requires some extra dependencies and is optional\n    pass\n\n\n\nNow you can interact with your bot! First, pick a thread to use as the key for this conversation.\n\nconfig = {"configurable": {"thread_id": "1"}}\n\nNext, call your chat bot.\n\nuser_input = "Hi there! My name is Will."\n\n# The config is the **second positional argument** to stream() or invoke()!\nevents = graph.stream(\n    {"messages": [{"role": "user", "content": user_input}]},\n    config,\n    stream_mode="values",\n)\nfor event in events:\n    event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nHi there! My name is Will.\n==================================[1m Ai Message [0m==================================\n\nHello Will! It\'s nice to meet you. How can I assist you today? Is there anything specific you\'d like to know or discuss?\n\nNote: The config was provided as the second positional argument when calling our graph. It importantly is not nested within the graph inputs ({\'messages\': []}).\n\nLet\'s ask a followup: see if it remembers your name.\n\nuser_input = "Remember my name?"\n\n# The config is the **second positional argument** to stream() or invoke()!\nevents = graph.stream(\n    {"messages": [{"role": "user", "content": user_input}]},\n    config,\n    stream_mode="values",\n)\nfor event in events:\n    event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nRemember my name?\n==================================[1m Ai Message [0m==================================\n\nOf course, I remember your name, Will. I always try to pay attention to important details that users share with me. Is there anything else you\'d like to talk about or any questions you have? I\'m here to help with a wide range of topics or tasks.\n\nNotice that we aren\'t using an external list for memory: it\'s all handled by the checkpointer! You can inspect the full execution in this LangSmith trace to see what\'s going on.\n\nDon\'t believe me? Try this using a different config.\n\n# The only difference is we change the `thread_id` here to "2" instead of "1"\nevents = graph.stream(\n    {"messages": [{"role": "user", "content": user_input}]},\n    {"configurable": {"thread_id": "2"}},\n    stream_mode="values",\n)\nfor event in events:\n    event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nRemember my name?\n==================================[1m Ai Message [0m==================================\n\nI apologize, but I don\'t have any previous context or memory of your name. As an AI assistant, I don\'t retain information from past conversations. Each interaction starts fresh. Could you please tell me your name so I can address you
```

properly in this conversation?\n\nNotice that the only change we\'ve made is to modify the thread_id in the config. See this call\'s LangSmith trace for comparison.\n\nBy now, we have made a few checkpoints across two different threads. But what goes into a checkpoint? To inspect a graph\'s state for a given config at any time, call get_state(config).\n\nsnapshot = graph.get_state(config)\nsnapshot\n\nStateSnapshot(values={\'messages\': [HumanMessage(content=\'Hi there! My name is Will.\', additional_kwargs={}, response_metadata={}, id=\'8c1ca919-c553-4ebf-95d4-b59a2d61e078\'), AIMessage(content="Hello Will! It\'s nice to meet you. How can I assist you today? Is there anything specific you\'d like to know or discuss?", additional_kwargs={}, response_metadata={\'id\': \'msg_01WTQebPhNwmMrmmWojJ9KXJ\', \'model\': \'claude-3-5-sonnet-20240620\', \'stop_reason\': \'end_turn\', \'stop_sequence\': None, \'usage\': {\'input_tokens\': 405, \'output_tokens\': 32}}, id=\'run-58587b77-8c82-41e6-8a90-d62c444a261d-0\', usage_metadata={\'input_tokens\': 405, \'output_tokens\': 32, \'total_tokens\': 437}), HumanMessage(content=\'Remember my name?\', additional_kwargs={}, response_metadata={}, id=\'daba7df6-ad75-4d6b-8057-745881cea1ca\'), AIMessage(content="Of course, I remember your name, Will. I always try to pay attention to important details that users share with me. Is there anything else you\'d like to talk about or any questions you have? I\'m here to help with a wide range of topics or tasks.", additional_kwargs={}, response_metadata={\'id\': \'msg_01E41KitY74HpENRgXx94vag\', \'model\': \'claude-3-5-sonnet-20240620\', \'stop_reason\': \'end_turn\', \'stop_sequence\': None, \'usage\': {\'input_tokens\': 444, \'output_tokens\': 58}}, id=\'run-ffeaae5c-4d2d-4ddb-bd59-5d5cbf2a5af8-0\', usage_metadata={\'input_tokens\': 444, \'output_tokens\': 58, \'total_tokens\': 502})]}, next=(), config={\'configurable\': {\'thread_id\': \'1\', \'checkpoint_ns\': \'\', \'checkpoint_id\': \'1ef7d06e-93e0-6acc-8004-f2ac846575d2\'}}, metadata={\'source\': \'loop\', \'writes\': {\'chatbot\': {\'messages\': [AIMessage(content="Of course, I remember your name, Will. I always try to pay attention to important details that users share with me. Is there anything else you\'d like to talk about or any questions you have? I\'m here to help with a wide range of topics or tasks.", additional_kwargs={}, response_metadata={\'id\': \'msg_01E41KitY74HpENRgXx94vag\', \'model\': \'claude-3-5-sonnet-20240620\', \'stop_reason\': \'end_turn\', \'stop_sequence\': None, \'usage\': {\'input_tokens\': 444, \'output_tokens\': 58}}, id=\'run-ffeaae5c-4d2d-4ddb-bd59-5d5cbf2a5af8-0\', usage_metadata={\'input_tokens\': 444, \'output_tokens\': 58, \'total_tokens\': 502})]}}, \'step\': 4, \'parents\': {}}, created_at=\'2024-09-27T19:30:10.820758+00:00\', parent_config={\'configurable\':

{\'thread_id\': \'1\', \'checkpoint_ns\': \'\', \'checkpoint_id\': \'1ef7d06e-859f-6206-8003-e1bd3c264b8f\'}}, tasks=())\n\nsnapshot.next # (since the graph ended this turn, `next` is empty. If you fetch a state from within a graph invocation, next tells which node will execute next)\n\n()\n\nThe snapshot above contains the current state values, corresponding config, and the next node to process. In our case, the graph has reached an END state, so next is empty.\n\nCongratulations! Your chatbot can now maintain conversation state across sessions thanks to LangGraph\'s checkpointing system. This opens up exciting possibilities for more natural, contextual interactions. LangGraph\'s checkpointing even handles arbitrarily complex graph states, which is much more expressive and powerful than simple chat memory.\n\nIn the next part, we\'ll introduce human oversight to our bot to handle situations where it may need guidance or verification before proceeding.\n\nCheck out the code snippet below to review our graph from this section.\n\nPart 4: Human-in-the-loop¶\n\nAgents can be unreliable and may need human input to successfully accomplish tasks. Similarly, for some actions, you may want to require human approval before running to ensure that everything is running as intended.\n\nLangGraph\'s persistence layer supports human-in-the-loop workflows, allowing execution to pause and resume based on user feedback. The primary interface to this functionality is the interrupt function. Calling interrupt inside a node will pause execution. Execution can be resumed, together with new input from a human, by passing in a Command. interrupt is ergonomically similar to Python\'s built-in input(), with some caveats. We demonstrate an example below.\n\nFirst, start with our existing code from Part 3. We will make one change, which is to add a simple human_assistance tool accessible to the chatbot. This tool uses interrupt to receive information from a human.\n\nfrom typing import Annotated\n\nfrom langchain_anthropic import ChatAnthropic\nfrom langchain_community.tools.tavily_search import TavilySearchResults\nfrom langchain_core.tools import tool\nfrom typing_extensions import TypedDict\n\nfrom langgraph.checkpoint.memory import MemorySaver\nfrom langgraph.graph import StateGraph, START, END\nfrom langgraph.graph.message import add_messages\nfrom langgraph.prebuilt import ToolNode, tools_condition\n\nfrom langgraph.types import Command, interrupt\n\n\nclass State(TypedDict):\n   messages: Annotated[list, add_messages]\n\n\ngraph_builder = StateGraph(State)\n\n\n@tool\ndef human_assistance(query: str) -> str:\n   """Request assistance from a human."""\n   human_response = interrupt({"query": query})\n   return human_response["data"]\n\n\ntool = TavilySearchResults(max_results=2)\ntools = [tool, human_assistance]\nllm = ChatAnthropic(model="claude-3-5-sonnet-20240620")\nllm_with_tools = llm.bind_tools(tools)\n\n\ndef chatbot(state: State):\n   message = llm_with_tools.invoke(state["messages"])\n   # Because we will be interrupting during tool execution,\n   # we disable parallel tool calling to avoid repeating any\n   # tool invocations when we resume.\n   assert len(message.tool_calls) <= 1\n

```
return {"messages": [message]}\n\n\ngraph_builder.add_node("chatbot",
chatbot)\n\ntool_node = ToolNode(tools=tools)\
ngraph_builder.add_node("tools", tool_node)\n\
ngraph_builder.add_conditional_edges(\n    "chatbot",\n
tools_condition,\n)\ngraph_builder.add_edge("tools", "chatbot")\
ngraph_builder.add_edge(START, "chatbot")\n\nAPI Reference:
ChatAnthropic | TavilySearchResults | tool | MemorySaver | StateGraph
| START | END | add_messages | ToolNode | tools_condition | Command |
interrupt\n\nTip\n\nCheck out the Human-in-the-loop section of the
How-to Guides for more examples of Human-in-the-loop workflows,
including how to review and edit tool calls before they are executed.\
n\nWe compile the graph with a checkpointer, as before:\n\nmemory =
MemorySaver()\n\ngraph = graph_builder.compile(checkpointer=memory)\n\
nVisualizing the graph, we recover the same layout as before. We have
just added a tool!\n\nfrom IPython.display import Image, display\n\
ntry:\n    display(Image(graph.get_graph().draw_mermaid_png()))\
nexcept Exception:\n    # This requires some extra dependencies and is
optional\n    pass\n\n\n\nLet\'s now prompt the chatbot with a
question that will engage the new human_assistance tool:\n\nuser_input
= "I need some expert guidance for building an AI agent. Could you
request assistance for me?"\nconfig = {"configurable": {"thread_id":
"1"}}\n\nevents = graph.stream(\n    {"messages": [{"role": "user",
"content": user_input}]},\n    config,\n    stream_mode="values",\n)\
nfor event in events:\n    if "messages" in event:\n
event["messages"][-1].pretty_print()\n\
n================================[1m Human Message
[0m=================================\n\nI need some expert guidance
for building an AI agent. Could you request assistance for me?\
n==================================[1m Ai Message
[0m==================================\n\n[{\'text\': "Certainly! I\'d
be happy to request expert assistance for you regarding building an AI
agent. To do this, I\'ll use the human_assistance function to relay
your request. Let me do that for you now.", \'type\': \'text\'},
{\'id\': \'toolu_01ABUqneqnuHNuo1vhfDFQCW\', \'input\':
{\'query\': \'A user is requesting expert guidance for building an AI
agent. Could you please provide some expert advice or resources on
this
topic?\'}, \'name\': \'human_assistance\', \'type\': \'tool_use\'}]\
nTool Calls:\n  human_assistance (toolu_01ABUqneqnuHNuo1vhfDFQCW)\n
Call ID: toolu_01ABUqneqnuHNuo1vhfDFQCW\n  Args:\n    query: A user is
requesting expert guidance for building an AI agent. Could you please
provide some expert advice or resources on this topic?\n\nThe chatbot
generated a tool call, but then execution has been interrupted! Note
that if we inspect the graph state, we see that it stopped at the
tools node:\n\nsnapshot = graph.get_state(config)\nsnapshot.next\n\
n(\'tools\',)\n\nLet\'s take a closer look at the human_assistance
tool:\n\n@tool\ndef human_assistance(query: str) -> str:\n
"""Request assistance from a human."""\n    human_response =
interrupt({"query": query})\n    return human_response["data"]\n\
nSimilar to Python\'s built-in input() function, calling interrupt
```

```
inside the tool will pause execution. Progress is persisted based on our choice of checkpointer-- so if we are persisting with Postgres, we can resume at any time as long as the database is alive. Here we are persisting with the in-memory checkpointer, so we can resume any time as long as our Python kernel is running.\n\nTo resume execution, we pass a Command object containing data expected by the tool. The format of this data can be customized based on our needs. Here, we just need a dict with a key "data":\n\nhuman_response = (\n    "We, the experts are here to help! We\'d recommend you check out LangGraph to build your agent."\n    " It\'s much more reliable and extensible than simple autonomous agents."\n)\n\nhuman_command = Command(resume={"data": human_response})\n\nevents = graph.stream(human_command, config, stream_mode="values")\nfor event in events:\n    if "messages" in event:\n        event["messages"][-1].pretty_print()\n\n==================================[1m Ai Message [0m==================================\n\n[{\'text\': "Certainly! I\'d be happy to request expert assistance for you regarding building an AI agent. To do this, I\'ll use the human_assistance function to relay your request. Let me do that for you now.", \'type\': \'text\'}, {\'id\': \'toolu_01ABUqneqnuHNuo1vhfDFQCW\', \'input\': {\'query\': \'A user is requesting expert guidance for building an AI agent. Could you please provide some expert advice or resources on this topic?\'}, \'name\': \'human_assistance\', \'type\': \'tool_use\'}]\nTool Calls:\n  human_assistance (toolu_01ABUqneqnuHNuo1vhfDFQCW)\n Call ID: toolu_01ABUqneqnuHNuo1vhfDFQCW\n  Args:\n    query: A user is requesting expert guidance for building an AI agent. Could you please provide some expert advice or resources on this topic?\n=================================[1m Tool Message [0m=================================\nName: human_assistance\n\nWe, the experts are here to help! We\'d recommend you check out LangGraph to build your agent. It\'s much more reliable and extensible than simple autonomous agents.\n==================================[1m Ai Message [0m==================================\n\nThank you for your patience. I\'ve received some expert advice regarding your request for guidance on building an AI agent. Here\'s what the experts have suggested:\n\nThe experts recommend that you look into LangGraph for building your AI agent. They mention that LangGraph is a more reliable and extensible option compared to simple autonomous agents.\n\nLangGraph is likely a framework or library designed specifically for creating AI agents with advanced capabilities. Here are a few points to consider based on this recommendation:\n\n1. Reliability: The experts emphasize that LangGraph is more reliable than simpler autonomous agent approaches. This could mean it has better stability, error handling, or consistent performance.\n\n2. Extensibility: LangGraph is described as more extensible, which suggests that it probably offers a flexible architecture that allows you to easily add new features or modify existing ones as your agent\'s requirements evolve.\n\n3. Advanced capabilities: Given that it\'s recommended over
```

"simple autonomous agents," LangGraph likely provides more sophisticated tools and techniques for building complex AI agents.\n\nTo get started with LangGraph, you might want to:\n\n1. Search for the official LangGraph documentation or website to learn more about its features and how to use it.\n2. Look for tutorials or guides specifically focused on building AI agents with LangGraph.\n3. Check if there are any community forums or discussion groups where you can ask questions and get support from other developers using LangGraph.\n\nIf you\'d like more specific information about LangGraph or have any questions about this recommendation, please feel free to ask, and I can request further assistance from the experts.\n\nOur input has been received and processed as a tool message. Review this call\'s LangSmith trace to see the exact work that was done in the above call. Notice that the state is loaded in the first step so that our chatbot can continue where it left off.\n\nCongrats! You\'ve used an interrupt to add human-in-the-loop execution to your chatbot, allowing for human oversight and intervention when needed. This opens up the potential UIs you can create with your AI systems. Since we have already added a checkpointer, as long as the underlying persistence layer is running, the graph can be paused indefinitely and resumed at any time as if nothing had happened.\n\nHuman-in-the-loop workflows enable a variety of new workflows and user experiences. Check out this section of the How-to Guides for more examples of Human-in-the-loop workflows, including how to review and edit tool calls before they are executed.\n\nPart 5: Customizing State¶\n\nSo far, we\'ve relied on a simple state with one entry-- a list of messages. You can go far with this simple state, but if you want to define complex behavior without relying on the message list, you can add additional fields to the state. Here we will demonstrate a new scenario, in which the chatbot is using its search tool to find specific information, and forwarding them to a human for review. Let\'s have the chatbot research the birthday of an entity. We will add name and birthday keys to the state:\n\nfrom typing import Annotated\n\nfrom typing_extensions import TypedDict\n\nfrom langgraph.graph.message import add_messages\n\n\nclass State(TypedDict):\n   messages: Annotated[list, add_messages]\n   name: str\n   birthday: str\n\nAPI Reference: add_messages\n\nAdding this information to the state makes it easily accessible by other graph nodes (e.g., a downstream node that stores or processes the information), as well as the graph\'s persistence layer.\n\nHere, we will populate the state keys inside of our human_assistance tool. This allows a human to review the information before it is stored in the state. We will again use Command, this time to issue a state update from inside our tool. Read more about use cases for Command here.\n\nfrom langchain_core.messages import ToolMessage\nfrom langchain_core.tools import InjectedToolCallId, tool\n\nfrom langgraph.types import Command, interrupt\n\n\n@tool\n# Note that because we are generating a ToolMessage for a state update, we\n# generally require the ID of the corresponding tool call. We can use\n# LangChain\'s InjectedToolCallId to signal that this argument

```
should not\n# be revealed to the model in the tool\'s schema.\ndef
human_assistance(\n    name: str, birthday: str, tool_call_id:
Annotated[str, InjectedToolCallId]\n) -> str:\n    """Request
assistance from a human."""\n    human_response = interrupt(\n
{\n            "question": "Is this correct?",\n            "name":
name,\n            "birthday": birthday,\n        },\n    )\n    # If
the information is correct, update the state as-is.\n    if
human_response.get("correct", "").lower().startswith("y"):\n
verified_name = name\n        verified_birthday = birthday\n
response = "Correct"\n    # Otherwise, receive information from the
human reviewer.\n    else:\n        verified_name =
human_response.get("name", name)\n        verified_birthday =
human_response.get("birthday", birthday)\n        response = f"Made a
correction: {human_response}"\n\n    # This time we explicitly update
the state with a ToolMessage inside\n    # the tool.\n    state_update
= {\n        "name": verified_name,\n        "birthday":
verified_birthday,\n        "messages": [ToolMessage(response,
tool_call_id=tool_call_id)],\n    }\n    # We return a Command object
in the tool to update our state.\n    return
Command(update=state_update)\n\nAPI Reference: ToolMessage |
InjectedToolCallId | tool | Command | interrupt\n\nOtherwise, the rest
of our graph is the same:\n\nfrom langchain_anthropic import
ChatAnthropic\nfrom langchain_community.tools.tavily_search import
TavilySearchResults\n\nfrom langgraph.checkpoint.memory import
MemorySaver\nfrom langgraph.graph import StateGraph, START, END\nfrom
langgraph.prebuilt import ToolNode, tools_condition\n\n\ntool =
TavilySearchResults(max_results=2)\ntools = [tool, human_assistance]\
nllm = ChatAnthropic(model="claude-3-5-sonnet-20240620")\
nllm_with_tools = llm.bind_tools(tools)\n\n\ndef chatbot(state:
State):\n    message = llm_with_tools.invoke(state["messages"])\n
assert len(message.tool_calls) <= 1\n    return {"messages":
[message]}\n\n\ngraph_builder = StateGraph(State)\
ngraph_builder.add_node("chatbot", chatbot)\n\ntool_node =
ToolNode(tools=tools)\ngraph_builder.add_node("tools", tool_node)\n\
ngraph_builder.add_conditional_edges(\n    "chatbot",\n
tools_condition,\n)\ngraph_builder.add_edge("tools", "chatbot")\
ngraph_builder.add_edge(START, "chatbot")\n\nmemory = MemorySaver()\
ngraph = graph_builder.compile(checkpointer=memory)\n\nAPI Reference:
ChatAnthropic | TavilySearchResults | MemorySaver | StateGraph | START
| END | ToolNode | tools_condition\n\nLet\'s prompt our application to
look up the "birthday" of the LangGraph library. We will direct the
chatbot to reach out to the human_assistance tool once it has the
required information. Note that setting name and birthday in the
arguments for the tool, we force the chatbot to generate proposals for
these fields.\n\nuser_input = (\n    "Can you look up when LangGraph
was released? "\n    "When you have the answer, use the
human_assistance tool for review."\n)\nconfig = {"configurable":
{"thread_id": "1"}}\n\nevents = graph.stream(\n    {"messages":
[{"role": "user", "content": user_input}]},\n    config,\n
```

stream_mode="values",\n)\nfor event in events:\n   if "messages" in event:\n   event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nCan you look up when LangGraph was released? When you have the answer, use the human_assistance tool for review.\n==================================[1m Ai Message [0m==================================\n\n[{\'text\': "Certainly! I\'ll start by searching for information about LangGraph\'s release date using the Tavily search function. Then, I\'ll use the human_assistance tool for review.", \'type\': \'text\'}, {\'id\': \'toolu_01JoXQPgTVJXiuma8xMVwqAi\', \'input\': {\'query\': \'LangGraph release date\'}, \'name\': \'tavily_search_results_json\', \'type\': \'tool_use\'}]\nTool Calls:\n tavily_search_results_json (toolu_01JoXQPgTVJXiuma8xMVwqAi)\n Call ID: toolu_01JoXQPgTVJXiuma8xMVwqAi\n Args:\n   query: LangGraph release date\n=================================[1m Tool Message [0m=================================\nName: tavily_search_results_json\n\n[{"url": "https://blog.langchain.dev/langgraph-cloud/", "content": "We also have a new stable release of LangGraph. By LangChain 6 min read Jun 27, 2024 (Oct \'24) Edit: Since the launch of LangGraph Cloud, we now have multiple deployment options alongside LangGraph Studio - which now fall under LangGraph Platform. LangGraph Cloud is synonymous with our Cloud SaaS deployment option."}, {"url": "https://changelog.langchain.com/announcements/langgraph-cloud-deploy-at-scale-monitor-carefully-iterate-boldly", "content": "LangChain - Changelog | ☁ LangGraph Cloud: Deploy at scale, monitor LangChain LangSmith LangGraph LangChain LangSmith LangGraph LangChain LangSmith LangGraph LangChain Changelog Sign up for our newsletter to stay up to date DATE: The LangChain Team LangGraph LangGraph Cloud ☁ LangGraph Cloud: Deploy at scale, monitor carefully, iterate boldly DATE: June 27, 2024 AUTHOR: The LangChain Team LangGraph Cloud is now in closed beta, offering scalable, fault-tolerant deployment for LangGraph agents. LangGraph Cloud also includes a new playground-like studio for debugging agent failure modes and quick iteration: Join the waitlist today for LangGraph Cloud. And to learn more, read our blog post announcement or check out our docs. Subscribe By clicking subscribe, you accept our privacy policy and terms and conditions."}]\n==================================[1m Ai Message [0m==================================\n\n[{\'text\': "Based on the search results, it appears that LangGraph was already in existence before June 27, 2024, when LangGraph Cloud was announced. However, the search results don\'t provide a specific release date for the original LangGraph. \\n\\nGiven this information, I\'ll use the human_assistance tool to review and potentially provide more accurate information about LangGraph\'s initial release date.", \'type\': \'text\'}, {\'id\': \'toolu_01JDQAV7nPqMkHHhNs3j3XoN\', \'input\':

{\'name\': \'Assistant\', \'birthday\': \'2023-01-01\'}, \'name\': \'human_assistance\', \'type\': \'tool_use\'}]\nTool Calls:\n human_assistance (toolu_01JDQAV7nPqMkHHhNs3j3XoN)\n Call ID: toolu_01JDQAV7nPqMkHHhNs3j3XoN\n Args:\n   name: Assistant\n   birthday: 2023-01-01\n\nWe\'ve hit the interrupt in the human_assistance tool again. In this case, the chatbot failed to identify the correct date, so we can supply it:\n\nhuman_command = Command(\n   resume={\n   "name": "LangGraph",\n   "birthday": "Jan 17, 2024",\n   },\n)\n\nevents = graph.stream(human_command, config, stream_mode="values")\nfor event in events:\n   if "messages" in event:\n   event["messages"][-1].pretty_print()\n\n==========[1m Ai Message [0m==========\n\n[{\'text\': "Based on the search results, it appears that LangGraph was already in existence before June 27, 2024, when LangGraph Cloud was announced. However, the search results don\'t provide a specific release date for the original LangGraph. \\n\\nGiven this information, I\'ll use the human_assistance tool to review and potentially provide more accurate information about LangGraph\'s initial release date.", \'type\': \'text\'},
{\'id\': \'toolu_01JDQAV7nPqMkHHhNs3j3XoN\', \'input\': {\'name\': \'Assistant\', \'birthday\': \'2023-01-01\'}, \'name\': \'human_assistance\', \'type\': \'tool_use\'}]\nTool Calls:\n human_assistance (toolu_01JDQAV7nPqMkHHhNs3j3XoN)\n Call ID: toolu_01JDQAV7nPqMkHHhNs3j3XoN\n Args:\n   name: Assistant\n   birthday: 2023-01-01\n==========[1m Tool Message [0m==========\nName: human_assistance\n\nMade a correction: {\'name\': \'LangGraph\', \'birthday\': \'Jan 17, 2024\'}\n==========[1m Ai Message [0m==========\n\nThank you for the human assistance. I can now provide you with the correct information about LangGraph\'s release date.\n\nLangGraph was initially released on January 17, 2024. This information comes from the human assistance correction, which is more accurate than the search results I initially found.\n\nTo summarize:\n1. LangGraph\'s original release date: January 17, 2024\n2. LangGraph Cloud announcement: June 27, 2024\n\nIt\'s worth noting that LangGraph had been in development and use for some time before the LangGraph Cloud announcement, but the official initial release of LangGraph itself was on January 17, 2024.\n\nNote that these fields are now reflected in the state:\n\nsnapshot = graph.get_state(config)\n\n{k: v for k, v in snapshot.values.items() if k in ("name", "birthday")}\n\n{\'name\': \'LangGraph\', \'birthday\': \'Jan 17, 2024\'}\n\nThis makes them easily accessible to downstream nodes (e.g., a node that further processes or stores the information).\n\nManually updating state¶\n\nLangGraph gives a high degree of control over the application state. For instance, at any point (including when interrupted), we can manually override a key using graph.update_state:\n\ngraph.update_state(config, {"name": "LangGraph

```
(library)"})\n\n{\'configurable\': {\'thread_id\': \'1\',\
n  \'checkpoint_ns\': \'\',\n  \'checkpoint_id\': \'1efd4ec5-cf69-
6352-8006-9278f1730162\'}}\n\nIf we call graph.get_state, we can see
the new value is reflected:\n\nsnapshot = graph.get_state(config)\n\
n{k: v for k, v in snapshot.values.items() if k in ("name",
"birthday")}\n\n{\'name\': \'LangGraph
(library)\', \'birthday\': \'Jan 17, 2024\'}\n\nManual state updates
will even generate a trace in LangSmith. If desired, they can also be
used to control human-in-the-loop workflows, as described in this
guide. Use of the interrupt function is generally recommended instead,
as it allows data to be transmitted in a human-in-the-loop interaction
independently of state updates.\n\nCongratulations! You\'ve added
custom keys to the state to facilitate a more complex workflow, and
learned how to generate state updates from inside tools.\n\nWe\'re
almost done with the tutorial, but there is one more concept we\'d
like to review before finishing that connects checkpointing and state
updates.\n\nThis section\'s code is reproduced below for your
reference.\n\nPart 6: Time Travel¶\n\nIn a typical chat bot workflow,
the user interacts with the bot 1 or more times to accomplish a task.
In the previous sections, we saw how to add memory and a human-in-the-
loop to be able to checkpoint our graph state and control future
responses.\n\nBut what if you want to let your user start from a
previous response and "branch off" to explore a separate outcome? Or
what if you want users to be able to "rewind" your assistant\'s work
to fix some mistakes or try a different strategy (common in
applications like autonomous software engineers)?\n\nYou can create
both of these experiences and more using LangGraph\'s built-in "time
travel" functionality.\n\nIn this section, you will "rewind" your
graph by fetching a checkpoint using the graph\'s get_state_history
method. You can then resume execution at this previous point in time.\
n\nFor this, let\'s use the simple chatbot with tools from Part 3:\n\
nfrom typing import Annotated\n\nfrom langchain_anthropic import
ChatAnthropic\nfrom langchain_community.tools.tavily_search import
TavilySearchResults\nfrom langchain_core.messages import BaseMessage\
nfrom typing_extensions import TypedDict\n\nfrom
langgraph.checkpoint.memory import MemorySaver\nfrom langgraph.graph
import StateGraph, START, END\nfrom langgraph.graph.message import
add_messages\nfrom langgraph.prebuilt import ToolNode,
tools_condition\n\n\nclass State(TypedDict):\n    messages:
Annotated[list, add_messages]\n\n\ngraph_builder = StateGraph(State)\
n\n\ntool = TavilySearchResults(max_results=2)\ntools = [tool]\nllm =
ChatAnthropic(model="claude-3-5-sonnet-20240620")\nllm_with_tools =
llm.bind_tools(tools)\n\n\ndef chatbot(state: State):\n    return
{"messages": [llm_with_tools.invoke(state["messages"])]}\n\n\
ngraph_builder.add_node("chatbot", chatbot)\n\ntool_node =
ToolNode(tools=[tool])\ngraph_builder.add_node("tools", tool_node)\n\
ngraph_builder.add_conditional_edges(\n    "chatbot",\n
tools_condition,\n)\ngraph_builder.add_edge("tools", "chatbot")\
ngraph_builder.add_edge(START, "chatbot")\n\nmemory = MemorySaver()\
```

ngraph = graph_builder.compile(checkpointer=memory)\n\nAPI Reference: ChatAnthropic | TavilySearchResults | BaseMessage | MemorySaver | StateGraph | START | END | add_messages | ToolNode | tools_condition\n\nLet\'s have our graph take a couple steps. Every step will be checkpointed in its state history:\n\nconfig = {"configurable": {"thread_id": "1"}}\nevents = graph.stream(\n   {\n   "messages": [\n   {\n   "role": "user",\n   "content": (\n   "I\'m learning LangGraph. "\n   "Could you do some research on it for me?"\n   ),\n   },\n   ],\n   },\n   config,\n   stream_mode="values",\n)\nfor event in events:\n   if "messages" in event:\n   event["messages"][-1].pretty_print()\n\n================================[1m Human Message [0m=================================\n\nI\'m learning LangGraph. Could you do some research on it for me?\n==================================[1m Ai Message [0m==================================\n\n[{\'text\': "Certainly! I\'d be happy to research LangGraph for you. To get the most up-to-date and accurate information, I\'ll use the Tavily search engine to look this up. Let me do that for you now.", \'type\': \'text\'}, {\'id\': \'toolu_01BscbfJJB9EWJFqGrN6E54e\', \'input\': {\'query\': \'LangGraph latest information and features\'}, \'name\': \'tavily_search_results_json\', \'type\': \'tool_use\'}]\nTool Calls:\n tavily_search_results_json (toolu_01BscbfJJB9EWJFqGrN6E54e)\n Call ID: toolu_01BscbfJJB9EWJFqGrN6E54e\n Args:\n   query: LangGraph latest information and features\n=================================[1m Tool Message [0m=================================\nName: tavily_search_results_json\n\n[{"url": "https://blockchain.news/news/langchain-new-features-upcoming-events-update", "content": "LangChain, a leading platform in the AI development space, has released its latest updates, showcasing new use cases and enhancements across its ecosystem. According to the LangChain Blog, the updates cover advancements in LangGraph Cloud, LangSmith\'s self-improving evaluators, and revamped documentation for LangGraph."}, {"url": "https://blog.langchain.dev/langgraph-platform-announce/", "content": "With these learnings under our belt, we decided to couple some of our latest offerings under LangGraph Platform. LangGraph Platform today includes LangGraph Server, LangGraph Studio, plus the CLI and SDK. ... we added features in LangGraph Server to deliver on a few key value areas. Below, we\'ll focus on these aspects of LangGraph Platform."}]\n==================================[1m Ai Message [0m==================================\n\nThank you for your patience. I\'ve found some recent information about LangGraph for you. Let me summarize the key points:\n\n1. LangGraph is part of the LangChain ecosystem, which is a leading platform in AI development.\n\n2. Recent updates and features of LangGraph include:\n\n   a. LangGraph Cloud: This seems to be a cloud-based version of LangGraph, though specific

```
details weren\'t provided in the search results.\n\n   b. LangGraph
Platform: This is a newly introduced concept that combines several
offerings:\n      - LangGraph Server\n      - LangGraph Studio\n
- CLI (Command Line Interface)\n      - SDK (Software Development
Kit)\n\n3. LangGraph Server: This component has received new features
to enhance its value proposition, though the specific features
weren\'t detailed in the search results.\n\n4. LangGraph Studio: This
appears to be a new tool in the LangGraph ecosystem, likely providing
a graphical interface for working with LangGraph.\n\n5. Documentation:
The LangGraph documentation has been revamped, which should make it
easier for learners like yourself to understand and use the tool.\n\
n6. Integration with LangSmith: While not directly part of LangGraph,
LangSmith (another tool in the LangChain ecosystem) now features self-
improving evaluators, which might be relevant if you\'re using
LangGraph as part of a larger LangChain project.\n\nAs you\'re
learning LangGraph, it would be beneficial to:\n\n1. Check out the
official LangChain documentation, especially the newly revamped
LangGraph sections.\n2. Explore the different components of the
LangGraph Platform (Server, Studio, CLI, and SDK) to see which best
fits your learning needs.\n3. Keep an eye on LangGraph Cloud
developments, as cloud-based solutions often provide an easier
starting point for learners.\n4. Consider how LangGraph fits into the
broader LangChain ecosystem, especially its interaction with tools
like LangSmith.\n\nIs there any specific aspect of LangGraph you\'d
like to know more about? I\'d be happy to do a more focused search on
particular features or use cases.\n\nevents = graph.stream(\n    {\n
"messages": [\n            {\n                "role": "user",\n
"content": (\n                    "Ya that\'s helpful. Maybe I\'ll "\n
"build an autonomous agent with it!"\n                ),\
n            },\n        ],\n    },\n    config,\n
stream_mode="values",\n)\nfor event in events:\n    if "messages" in
event:\n        event["messages"][-1].pretty_print()\n\
n================================[1m Human Message
[0m=================================\n\nYa that\'s helpful. Maybe
I\'ll build an autonomous agent with it!\
n==================================[1m Ai Message
[0m==================================\n\n[{\'text\': "That\'s an
exciting idea! Building an autonomous agent with LangGraph is indeed a
great application of this technology. LangGraph is particularly well-
suited for creating complex, multi-step AI workflows, which is perfect
for autonomous agents. Let me gather some more specific information
about using LangGraph for building autonomous
agents.", \'type\': \'text\'},
{\'id\': \'toolu_01QWNHhUaeeWcGXvA4eHT7Zo\', \'input\':
{\'query\': \'Building autonomous agents with LangGraph examples and
tutorials\'}, \'name\': \'tavily_search_results_json\', \'type\': \'to
ol_use\'}]\nTool Calls:\n  tavily_search_results_json
(toolu_01QWNHhUaeeWcGXvA4eHT7Zo)\n Call ID:
toolu_01QWNHhUaeeWcGXvA4eHT7Zo\n  Args:\n    query: Building
```

autonomous agents with LangGraph examples and tutorials\
n==========[1m Tool Message
[0m==========\nName:
tavily_search_results_json\n\n[{"url":
"https://towardsdatascience.com/building-autonomous-multi-tool-agents-
with-gemini-2-0-and-langgraph-ad3d7bd5e79d", "content": "Building
Autonomous Multi-Tool Agents with Gemini 2.0 and LangGraph | by
Youness Mansar | Jan, 2025 | Towards Data Science Building Autonomous
Multi-Tool Agents with Gemini 2.0 and LangGraph A practical tutorial
with full code examples for building and running multi-tool agents
Towards Data Science LLMs are remarkable – they can memorize vast
amounts of information, answer general knowledge questions, write
code, generate stories, and even fix your grammar. In this tutorial,
we are going to build a simple LLM agent that is equipped with four
tools that it can use to answer a user's question. This Agent will
have the following specifications: Follow Published in Towards Data
Science ---------- Your home for data science
and AI. Follow Follow Follow"}, {"url":
"https://github.com/anmolaman20/Tools_and_Agents", "content": "GitHub
- anmolaman20/Tools_and_Agents: This repository provides resources for
building AI agents using Langchain and Langgraph. This repository
provides resources for building AI agents using Langchain and
Langgraph. This repository provides resources for building AI agents
using Langchain and Langgraph. This repository serves as a
comprehensive guide for building AI-powered agents using Langchain and
Langgraph. It provides hands-on examples, practical tutorials, and
resources for developers and AI enthusiasts to master building
intelligent systems and workflows. AI Agent Development: Gain insights
into creating intelligent systems that think, reason, and adapt in
real time. This repository is ideal for AI practitioners, developers
exploring language models, or anyone interested in building
intelligent systems. This repository provides resources for building
AI agents using Langchain and Langgraph."}]\
n==========[1m Ai Message
[0m==========\n\nGreat idea! Building an
autonomous agent with LangGraph is definitely an exciting project.
Based on the latest information I\'ve found, here are some insights
and tips for building autonomous agents with LangGraph:\n\n1. Multi-
Tool Agents: LangGraph is particularly well-suited for creating
autonomous agents that can use multiple tools. This allows your agent
to have a diverse set of capabilities and choose the right tool for
each task.\n\n2. Integration with Large Language Models (LLMs): You
can combine LangGraph with powerful LLMs like Gemini 2.0 to create
more intelligent and capable agents. The LLM can serve as the "brain"
of your agent, making decisions and generating responses.\n\n3.
Workflow Management: LangGraph excels at managing complex, multi-step
AI workflows. This is crucial for autonomous agents that need to break
down tasks into smaller steps and execute them in the right order.\n\
n4. Practical Tutorials Available: There are tutorials available that

provide full code examples for building and running multi-tool agents. These can be incredibly helpful as you start your project.\n\n5. Langchain Integration: LangGraph is often used in conjunction with Langchain. This combination provides a powerful framework for building AI agents, offering features like memory management, tool integration, and prompt management.\n\n6. GitHub Resources: There are repositories available (like the one by anmolaman20) that provide comprehensive resources for building AI agents using Langchain and LangGraph. These can be valuable references as you develop your agent.\n\n7. Real-time Adaptation: LangGraph allows you to create agents that can think, reason, and adapt in real-time, which is crucial for truly autonomous behavior.\n\n8. Customization: You can equip your agent with specific tools tailored to your use case. For example, you might include tools for web searching, data analysis, or interacting with specific APIs.\n\nTo get started with your autonomous agent project:\n\n1. Familiarize yourself with LangGraph\'s documentation and basic concepts.\n2. Look into tutorials that specifically deal with building autonomous agents, like the one mentioned from Towards Data Science.\n3. Decide on the specific capabilities you want your agent to have and identify the tools it will need.\n4. Start with a simple agent and gradually add complexity as you become more comfortable with the framework.\n5. Experiment with different LLMs to find the one that works best for your use case.\n6. Pay attention to how you structure the agent\'s decision-making process and workflow.\n7. Don\'t forget to implement proper error handling and safety measures, especially if your agent will be interacting with external systems or making important decisions.\n\nBuilding an autonomous agent is an iterative process, so be prepared to refine and improve your agent over time. Good luck with your project! If you need any more specific information as you progress, feel free to ask.\n\nNow that we\'ve had the agent take a couple steps, we can replay the full state history to see everything that occurred.\n\nto_replay = None\nfor state in graph.get_state_history(config):\n   print("Num Messages: ", len(state.values["messages"]), "Next: ", state.next)\n   print("-" * 80)\n   if len(state.values["messages"]) == 6:\n   # We are somewhat arbitrarily selecting a specific state based on the number of chat messages in the state.\n   to_replay = state\n\nNum Messages: 8 Next: ()\n
n--\n----------\nNum Messages: 7 Next: (\'chatbot\',)\n
n--\n----------\nNum Messages: 6 Next: (\'tools\',)\n
n--\n----------\nNum Messages: 5 Next: (\'chatbot\',)\n
n--\n----------\nNum Messages: 4 Next: (\'__start__\',)\n
n--\n----------\nNum Messages: 4 Next: ()\n
n--

```
----------\nNum Messages:  3 Next:  (\'chatbot\',)\
n----------\nNum Messages:  2 Next:  (\'tools\',)\
n----------\nNum Messages:  1 Next:  (\'chatbot\',)\
n----------\nNum Messages:  0 Next:  (\'__start__\',)\
n----------\n\nNotice that checkpoints are saved for every step of the
graph. This spans invocations so you can rewind across a full
thread\'s history. We\'ve picked out to_replay as a state to resume
from. This is the state after the chatbot node in the second graph
invocation above.\n\nResuming from this point should call the action
node next.\n\nprint(to_replay.next)\nprint(to_replay.config)\n\
n(\'tools\',)\n{\'configurable\':
{\'thread_id\': \'1\', \'checkpoint_ns\': \'\', \'checkpoint_id\': \'1
efd43e3-0c1f-6c4e-8006-891877d65740\'}}\n\nNotice that the
checkpoint\'s config (to_replay.config) contains a checkpoint_id
timestamp. Providing this checkpoint_id value tells LangGraph\'s
checkpointer to load the state from that moment in time. Let\'s try it
below:\n\n# The `checkpoint_id` in the `to_replay.config` corresponds
to a state we\'ve persisted to our checkpointer.\nfor event in
graph.stream(None, to_replay.config, stream_mode="values"):\n    if
"messages" in event:\n        event["messages"][-1].pretty_print()\n\
n================================[1m Ai Message
[0m==================================\n\n[{\'text\': "That\'s an
exciting idea! Building an autonomous agent with LangGraph is indeed a
great application of this technology. LangGraph is particularly well-
suited for creating complex, multi-step AI workflows, which is perfect
for autonomous agents. Let me gather some more specific information
about using LangGraph for building autonomous
agents.", \'type\': \'text\'},
{\'id\': \'toolu_01QWNHhUaeeWcGXvA4eHT7Zo\', \'input\':
{\'query\': \'Building autonomous agents with LangGraph examples and
tutorials\'}, \'name\': \'tavily_search_results_json\', \'type\': \'to
ol_use\'}]\nTool Calls:\n  tavily_search_results_json
(toolu_01QWNHhUaeeWcGXvA4eHT7Zo)\n Call ID:
toolu_01QWNHhUaeeWcGXvA4eHT7Zo\n  Args:\n    query: Building
autonomous agents with LangGraph examples and tutorials\
n=================================[1m Tool Message
[0m=================================\nName:
tavily_search_results_json\n\n[{"url":
"https://towardsdatascience.com/building-autonomous-multi-tool-agents-
with-gemini-2-0-and-langgraph-ad3d7bd5e79d", "content": "Building
Autonomous Multi-Tool Agents with Gemini 2.0 and LangGraph | by
Youness Mansar | Jan, 2025 | Towards Data Science Building Autonomous
Multi-Tool Agents with Gemini 2.0 and LangGraph A practical tutorial
with full code examples for building and running multi-tool agents
Towards Data Science LLMs are remarkable — they can memorize vast
amounts of information, answer general knowledge questions, write
```

code, generate stories, and even fix your grammar. In this tutorial, we are going to build a simple LLM agent that is equipped with four tools that it can use to answer a user’s question. This Agent will have the following specifications: Follow Published in Towards Data Science ---------- Your home for data science and AI. Follow Follow Follow"}, {"url": "https://github.com/anmolaman20/Tools_and_Agents", "content": "GitHub - anmolaman20/Tools_and_Agents: This repository provides resources for building AI agents using Langchain and Langgraph. This repository provides resources for building AI agents using Langchain and Langgraph. This repository provides resources for building AI agents using Langchain and Langgraph. This repository serves as a comprehensive guide for building AI-powered agents using Langchain and Langgraph. It provides hands-on examples, practical tutorials, and resources for developers and AI enthusiasts to master building intelligent systems and workflows. AI Agent Development: Gain insights into creating intelligent systems that think, reason, and adapt in real time. This repository is ideal for AI practitioners, developers exploring language models, or anyone interested in building intelligent systems. This repository provides resources for building AI agents using Langchain and Langgraph."}]\n==========[1m Ai Message [0m==========\n\nGreat idea! Building an autonomous agent with LangGraph is indeed an excellent way to apply and deepen your understanding of the technology. Based on the search results, I can provide you with some insights and resources to help you get started:\n\n1. Multi-Tool Agents:\n   LangGraph is well-suited for building autonomous agents that can use multiple tools. This allows your agent to have a variety of capabilities and choose the appropriate tool based on the task at hand.\n\n2. Integration with Large Language Models (LLMs):\n   There\'s a tutorial that specifically mentions using Gemini 2.0 (Google\'s LLM) with LangGraph to build autonomous agents. This suggests that LangGraph can be integrated with various LLMs, giving you flexibility in choosing the language model that best fits your needs.\n\n3. Practical Tutorials:\n   There are tutorials available that provide full code examples for building and running multi-tool agents. These can be invaluable as you start your project, giving you a concrete starting point and demonstrating best practices.\n\n4. GitHub Resources:\n   There\'s a GitHub repository (github.com/anmolaman20/Tools_and_Agents) that provides resources for building AI agents using both Langchain and Langgraph. This could be a great resource for code examples, tutorials, and understanding how LangGraph fits into the broader LangChain ecosystem.\n\n5. Real-Time Adaptation:\n   The resources mention creating intelligent systems that can think, reason, and adapt in real-time. This is a key feature of advanced autonomous agents and something you can aim for in your project.\n\n6. Diverse Applications:\n   The materials suggest that these techniques can be applied to various tasks, from answering questions to potentially more

```
complex decision-making processes.\n\nTo get started with your
autonomous agent project using LangGraph, you might want to:\n\n1.
Review the tutorials mentioned, especially those with full code
examples.\n2. Explore the GitHub repository for hands-on examples and
resources.\n3. Decide on the specific tasks or capabilities you want
your agent to have.\n4. Choose an LLM to integrate with LangGraph
(like GPT, Gemini, or others).\n5. Start with a simple agent that uses
one or two tools, then gradually expand its capabilities.\n6.
Implement decision-making logic to help your agent choose between
different tools or actions.\n7. Test your agent thoroughly with
various inputs and scenarios to ensure robust performance.\n\
nRemember, building an autonomous agent is an iterative process. Start
simple and gradually increase complexity as you become more
comfortable with LangGraph and its capabilities.\n\nWould you like
more information on any specific aspect of building your autonomous
agent with LangGraph?\n\nNotice that the graph resumed execution from
the **action** node. You can tell this is the case since the first
value printed above is the response from our search engine tool.\n\
nCongratulations! You\'ve now used time-travel checkpoint traversal in
LangGraph. Being able to rewind and explore alternative paths opens up
a world of possibilities for debugging, experimentation, and
interactive applications.\n\nNext Steps¶\n\nTake your journey further
by exploring deployment and advanced features:\n\nServer Quickstart¶\
n\nLangGraph Server Quickstart: Launch a LangGraph server locally and
interact with it using the REST API and LangGraph Studio Web UI.\n\
nLangGraph Cloud¶\n\nLangGraph Cloud QuickStart: Deploy your LangGraph
app using LangGraph Cloud.\n\nLangGraph Framework¶\n\nLangGraph
Concepts: Learn the foundational concepts of LangGraph.\n\nLangGraph
How-to Guides: Guides for common tasks with LangGraph.\n\nLangGraph
Platform¶\n\nExpand your knowledge with these resources:\n\nLangGraph
Platform Concepts: Understand the foundational concepts of the
LangGraph Platform.\n\nLangGraph Platform How-to Guides: Guides for
common tasks with LangGraph Platform.\n\nComments')]
```

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
text_splitter=RecursiveCharacterTextSplitter(chunk_size=1000,chunk_ove
rlap=200)
all_splits=text_splitter.split_documents(docs)
print("Total number of documents",len(all_splits))
```

```
Total number of documents 98
```

```
all_splits[7]
```

```
Document(metadata={'source':
'https://langchain-ai.github.io/langgraph/tutorials/introduction/'},
page_content="Assistant: LangGraph is a library designed to help build
stateful multi-agent applications using language models. It provides
tools for creating workflows and state machines to coordinate multiple
AI agents or language model interactions. LangGraph is built on top of
```

```
LangChain, leveraging its components while adding graph-based coordination capabilities. It's particularly useful for developing more complex, stateful AI applications that go beyond simple query-response interactions.\nGoodbye!\n\nCongratulations! You've built your first chatbot using LangGraph. This bot can engage in basic conversation by taking user input and generating responses using an LLM. You can inspect a LangSmith Trace for the call above at the provided link.\n\nHowever, you may have noticed that the bot's knowledge is limited to what's in its training data. In the next part, we'll add a web search tool to expand the bot's knowledge and make it more capable.")
```

```
from langchain_community.embeddings import HuggingFaceBgeEmbeddings
embeddings=HuggingFaceBgeEmbeddings()
vector=embeddings.embed_query("Hello world")
vector[:5]
```

```
<ipython-input-8-b6f1168fc35b>:2: LangChainDeprecationWarning: The class `HuggingFaceBgeEmbeddings` was deprecated in LangChain 0.2.2 and will be removed in 1.0. An updated version of the class exists in the :class:`~langchain-huggingface package and should be used instead. To use it run `pip install -U :class:`~langchain-huggingface` and import as `from :class:`~langchain_huggingface import HuggingFaceEmbeddings``.
  embeddings=HuggingFaceBgeEmbeddings()
<ipython-input-8-b6f1168fc35b>:2: LangChainDeprecationWarning: Default values for HuggingFaceBgeEmbeddings.model_name were deprecated in LangChain 0.2.5 and will be removed in 0.4.0. Explicitly pass a model_name to the HuggingFaceBgeEmbeddings constructor instead.
  embeddings=HuggingFaceBgeEmbeddings()
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```


```
{"model_id":"e48db198e7dd4db8b93a2b43e87b92ba","version_major":2,"version_minor":0}
```

```
{"model_id":"47a81788b1564c118ed08d89fdc4d808","version_major":2,"version_minor":0}
```

```
{"model_id":"464b2e41eb7f450abf7835dc77f3f8df","version_major":2,"version_minor":0}
```

{"model_id":"7da714edef3e42e184eb602dac91d1a2","version_major":2,"version_minor":0}

{"model_id":"552c345622364f0780cb422e12465d5c","version_major":2,"version_minor":0}

{"model_id":"fa133b74c4d748148aa81746f13f6790","version_major":2,"version_minor":0}

{"model_id":"79c181d8ee224c358ea8bde2cd18fd5d","version_major":2,"version_minor":0}

{"model_id":"0fbe08ab2c66485f874700b814473afd","version_major":2,"version_minor":0}

{"model_id":"af46f8fefdfd493593d7563a8100a758","version_major":2,"version_minor":0}

{"model_id":"b118d305ffce49a4bdcf5e6e08e3d589","version_major":2,"version_minor":0}

{"model_id":"b068affe1f88484fbe885b5315878f1c","version_major":2,"version_minor":0}

```
[0.022817175835371017,
 -0.011265435256063938,
 0.007570384070277214,
 -0.010442364029586315,
 -0.02732078544795513]
```

```
pip install langchain_chroma
```

```
Requirement already satisfied: langchain_chroma in
/usr/local/lib/python3.11/dist-packages (0.2.2)
Requirement already satisfied: langchain-core!=0.3.0,!=0.3.1,!
=0.3.10,!=0.3.11,!=0.3.12,!=0.3.13,!=0.3.14,!=0.3.2,!=0.3.3,!=0.3.4,!
=0.3.5,!=0.3.6,!=0.3.7,!=0.3.8,!=0.3.9,<0.4.0,>=0.2.43 in
/usr/local/lib/python3.11/dist-packages (from langchain_chroma)
(0.3.51)
Requirement already satisfied: numpy<2.0.0,>=1.22.4 in
/usr/local/lib/python3.11/dist-packages (from langchain_chroma)
(1.26.4)
Requirement already satisfied: chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from langchain_chroma)
(0.6.3)
Requirement already satisfied: build>=1.0.3 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (1.2.2.post1)
Requirement already satisfied: pydantic>=1.9 in
```

```
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (2.11.1)
Requirement already satisfied: chroma-hnswlib==0.7.6 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (0.7.6)
Requirement already satisfied: fastapi>=0.95.2 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (0.115.12)
Requirement already satisfied: uvicorn>=0.18.3 in
/usr/local/lib/python3.11/dist-packages (from
uvicorn[standard]>=0.18.3->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma)
(0.34.0)
Requirement already satisfied: posthog>=2.4.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (3.23.0)
Requirement already satisfied: typing_extensions>=4.5.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (4.13.0)
Requirement already satisfied: onnxruntime>=1.14.1 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (1.21.0)
Requirement already satisfied: opentelemetry-api>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (1.31.1)
Requirement already satisfied: opentelemetry-exporter-otlp-proto-
grpc>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.31.1)
Requirement already satisfied: opentelemetry-instrumentation-
fastapi>=0.41b0 in /usr/local/lib/python3.11/dist-packages (from
chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (0.52b1)
Requirement already satisfied: opentelemetry-sdk>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (1.31.1)
Requirement already satisfied: tokenizers>=0.13.2 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (0.21.1)
Requirement already satisfied: pypika>=0.48.9 in
```

```
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (0.48.9)
Requirement already satisfied: tqdm>=4.65.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (4.67.1)
Requirement already satisfied: overrides>=7.3.1 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (7.7.0)
Requirement already satisfied: importlib-resources in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (6.5.2)
Requirement already satisfied: grpcio>=1.58.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (1.71.0)
Requirement already satisfied: bcrypt>=4.0.1 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (4.3.0)
Requirement already satisfied: typer>=0.9.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (0.15.2)
Requirement already satisfied: kubernetes>=28.1.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (32.0.1)
Requirement already satisfied: tenacity>=8.2.3 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (9.1.2)
Requirement already satisfied: PyYAML>=6.0.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (6.0.2)
Requirement already satisfied: mmh3>=4.0.1 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (5.1.0)
Requirement already satisfied: orjson>=3.9.12 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (3.10.16)
Requirement already satisfied: httpx>=0.27.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
```

```
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (0.28.1)
Requirement already satisfied: rich>=10.11.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (13.9.4)
Requirement already satisfied: langsmith<0.4,>=0.1.125 in
/usr/local/lib/python3.11/dist-packages (from langchain-core!=0.3.0,!
=0.3.1,!=0.3.10,!=0.3.11,!=0.3.12,!=0.3.13,!=0.3.14,!=0.3.2,!=0.3.3,!
=0.3.4,!=0.3.5,!=0.3.6,!=0.3.7,!=0.3.8,!=0.3.9,<0.4.0,>=0.2.43-
>langchain_chroma) (0.3.22)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in
/usr/local/lib/python3.11/dist-packages (from langchain-core!=0.3.0,!
=0.3.1,!=0.3.10,!=0.3.11,!=0.3.12,!=0.3.13,!=0.3.14,!=0.3.2,!=0.3.3,!
=0.3.4,!=0.3.5,!=0.3.6,!=0.3.7,!=0.3.8,!=0.3.9,<0.4.0,>=0.2.43-
>langchain_chroma) (1.33)
Requirement already satisfied: packaging<25,>=23.2 in
/usr/local/lib/python3.11/dist-packages (from langchain-core!=0.3.0,!
=0.3.1,!=0.3.10,!=0.3.11,!=0.3.12,!=0.3.13,!=0.3.14,!=0.3.2,!=0.3.3,!
=0.3.4,!=0.3.5,!=0.3.6,!=0.3.7,!=0.3.8,!=0.3.9,<0.4.0,>=0.2.43-
>langchain_chroma) (24.2)
Requirement already satisfied: pyproject_hooks in
/usr/local/lib/python3.11/dist-packages (from build>=1.0.3->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.2.0)
Requirement already satisfied: starlette<0.47.0,>=0.40.0 in
/usr/local/lib/python3.11/dist-packages (from fastapi>=0.95.2-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (0.46.1)
Requirement already satisfied: anyio in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.27.0->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (4.9.0)
Requirement already satisfied: certifi in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.27.0->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (2025.1.31)
Requirement already satisfied: httpcore==1.* in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.27.0->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.0.7)
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-
packages (from httpx>=0.27.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma)
(3.10)
Requirement already satisfied: h11<0.15,>=0.13 in
/usr/local/lib/python3.11/dist-packages (from httpcore==1.*-
>httpx>=0.27.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!
=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (0.14.0)
```

```
Requirement already satisfied: jsonpointer>=1.9 in
/usr/local/lib/python3.11/dist-packages (from jsonpatch<2.0,>=1.33-
>langchain-core!=0.3.0,!=0.3.1,!=0.3.10,!=0.3.11,!=0.3.12,!=0.3.13,!
=0.3.14,!=0.3.2,!=0.3.3,!=0.3.4,!=0.3.5,!=0.3.6,!=0.3.7,!=0.3.8,!
=0.3.9,<0.4.0,>=0.2.43->langchain_chroma) (3.0.0)
Requirement already satisfied: six>=1.9.0 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.17.0)
Requirement already satisfied: python-dateutil>=2.5.3 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (2.8.2)
Requirement already satisfied: google-auth>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (2.38.0)
Requirement already satisfied: websocket-client!=0.40.0,!=0.41.*,!
=0.42.*,>=0.32.0 in /usr/local/lib/python3.11/dist-packages (from
kubernetes>=28.1.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!
=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.8.0)
Requirement already satisfied: requests in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (2.32.3)
Requirement already satisfied: requests-oauthlib in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (2.0.0)
Requirement already satisfied: oauthlib>=3.2.2 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (3.2.2)
Requirement already satisfied: urllib3>=1.24.2 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (2.3.0)
Requirement already satisfied: durationpy>=0.7 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (0.9)
Requirement already satisfied: requests-toolbelt<2.0.0,>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.125-
>langchain-core!=0.3.0,!=0.3.1,!=0.3.10,!=0.3.11,!=0.3.12,!=0.3.13,!
=0.3.14,!=0.3.2,!=0.3.3,!=0.3.4,!=0.3.5,!=0.3.6,!=0.3.7,!=0.3.8,!
=0.3.9,<0.4.0,>=0.2.43->langchain_chroma) (1.0.0)
Requirement already satisfied: zstandard<0.24.0,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.125-
>langchain-core!=0.3.0,!=0.3.1,!=0.3.10,!=0.3.11,!=0.3.12,!=0.3.13,!
```

```
=0.3.14,!=0.3.2,!=0.3.3,!=0.3.4,!=0.3.5,!=0.3.6,!=0.3.7,!=0.3.8,!
=0.3.9,<0.4.0,>=0.2.43->langchain_chroma) (0.23.0)
Requirement already satisfied: coloredlogs in
/usr/local/lib/python3.11/dist-packages (from onnxruntime>=1.14.1-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (15.0.1)
Requirement already satisfied: flatbuffers in
/usr/local/lib/python3.11/dist-packages (from onnxruntime>=1.14.1-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (25.2.10)
Requirement already satisfied: protobuf in
/usr/local/lib/python3.11/dist-packages (from onnxruntime>=1.14.1-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (5.29.4)
Requirement already satisfied: sympy in
/usr/local/lib/python3.11/dist-packages (from onnxruntime>=1.14.1-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.13.1)
Requirement already satisfied: deprecated>=1.2.6 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
api>=1.2.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!
=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.2.18)
Requirement already satisfied: importlib-metadata<8.7.0,>=6.0 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
api>=1.2.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!
=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (8.6.1)
Requirement already satisfied: googleapis-common-protos~=1.52 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-exporter-
otlp-proto-grpc>=1.2.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!
=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.69.2)
Requirement already satisfied: opentelemetry-exporter-otlp-proto-
common==1.31.1 in /usr/local/lib/python3.11/dist-packages (from
opentelemetry-exporter-otlp-proto-grpc>=1.2.0->chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (1.31.1)
Requirement already satisfied: opentelemetry-proto==1.31.1 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-exporter-
otlp-proto-grpc>=1.2.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!
=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.31.1)
Requirement already satisfied: opentelemetry-instrumentation-
asgi==0.52b1 in /usr/local/lib/python3.11/dist-packages (from
opentelemetry-instrumentation-fastapi>=0.41b0->chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (0.52b1)
Requirement already satisfied: opentelemetry-instrumentation==0.52b1
in /usr/local/lib/python3.11/dist-packages (from opentelemetry-
instrumentation-fastapi>=0.41b0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma)
(0.52b1)
```

```
Requirement already satisfied: opentelemetry-semantic-
conventions==0.52b1 in /usr/local/lib/python3.11/dist-packages (from
opentelemetry-instrumentation-fastapi>=0.41b0->chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (0.52b1)
Requirement already satisfied: opentelemetry-util-http==0.52b1 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
instrumentation-fastapi>=0.41b0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma)
(0.52b1)
Requirement already satisfied: wrapt<2.0.0,>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
instrumentation==0.52b1->opentelemetry-instrumentation-
fastapi>=0.41b0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!
=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.17.2)
Requirement already satisfied: asgiref~=3.0 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
instrumentation-asgi==0.52b1->opentelemetry-instrumentation-
fastapi>=0.41b0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!
=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (3.8.1)
Requirement already satisfied: monotonic>=1.5 in
/usr/local/lib/python3.11/dist-packages (from posthog>=2.4.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.6)
Requirement already satisfied: backoff>=1.10.0 in
/usr/local/lib/python3.11/dist-packages (from posthog>=2.4.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (2.2.1)
Requirement already satisfied: distro>=1.5.0 in
/usr/local/lib/python3.11/dist-packages (from posthog>=2.4.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.9.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=1.9->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=1.9->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (2.33.0)
Requirement already satisfied: typing-inspection>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=1.9->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (0.4.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (2.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in
/usr/local/lib/python3.11/dist-packages (from tokenizers>=0.13.2-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (0.30.1)
Requirement already satisfied: click>=8.0.0 in
/usr/local/lib/python3.11/dist-packages (from typer>=0.9.0->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in
/usr/local/lib/python3.11/dist-packages (from typer>=0.9.0->chromadb!
=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.5.4)
Requirement already satisfied: httptools>=0.6.3 in
/usr/local/lib/python3.11/dist-packages (from
uvicorn[standard]>=0.18.3->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma)
(0.6.4)
Requirement already satisfied: python-dotenv>=0.13 in
/usr/local/lib/python3.11/dist-packages (from
uvicorn[standard]>=0.18.3->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma)
(1.1.0)
Requirement already satisfied: uvloop!=0.15.0,!=0.15.1,>=0.14.0 in
/usr/local/lib/python3.11/dist-packages (from
uvicorn[standard]>=0.18.3->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma)
(0.21.0)
Requirement already satisfied: watchfiles>=0.13 in
/usr/local/lib/python3.11/dist-packages (from
uvicorn[standard]>=0.18.3->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma)
(1.0.4)
Requirement already satisfied: websockets>=10.4 in
/usr/local/lib/python3.11/dist-packages (from
uvicorn[standard]>=0.18.3->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!
=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma)
(15.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.11/dist-packages (from google-auth>=1.0.1-
>kubernetes>=28.1.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!
=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (5.5.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.11/dist-packages (from google-auth>=1.0.1-
>kubernetes>=28.1.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!
=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (0.4.2)
Requirement already satisfied: rsa<5,>=3.1.4 in
```

```
/usr/local/lib/python3.11/dist-packages (from google-auth>=1.0.1-
>kubernetes>=28.1.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!
=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (4.9)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.16.4->tokenizers>=0.13.2->chromadb!=0.5.10,!=0.5.11,!
=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (3.18.0)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.16.4->tokenizers>=0.13.2->chromadb!=0.5.10,!=0.5.11,!
=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (2025.3.2)
Requirement already satisfied: zipp>=3.20 in
/usr/local/lib/python3.11/dist-packages (from importlib-
metadata<8.7.0,>=6.0->opentelemetry-api>=1.2.0->chromadb!=0.5.10,!
=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (3.21.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0-
>rich>=10.11.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!
=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (0.1.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests-
>kubernetes>=28.1.0->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!
=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (3.4.1)
Requirement already satisfied: sniffio>=1.1 in
/usr/local/lib/python3.11/dist-packages (from anyio->httpx>=0.27.0-
>chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!
=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.3.1)
Requirement already satisfied: humanfriendly>=9.1 in
/usr/local/lib/python3.11/dist-packages (from coloredlogs-
>onnxruntime>=1.14.1->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!
=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (10.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from sympy-
>onnxruntime>=1.14.1->chromadb!=0.5.10,!=0.5.11,!=0.5.12,!=0.5.4,!
=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0->langchain_chroma) (1.3.0)
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in
/usr/local/lib/python3.11/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth>=1.0.1->kubernetes>=28.1.0->chromadb!=0.5.10,!=0.5.11,!
=0.5.12,!=0.5.4,!=0.5.5,!=0.5.7,!=0.5.9,<0.7.0,>=0.4.0-
>langchain_chroma) (0.6.1)

pip install langchain langchain-community chromadb sentence-
transformers

Requirement already satisfied: langchain in
/usr/local/lib/python3.11/dist-packages (0.3.23)
Requirement already satisfied: langchain-community in
```

```
/usr/local/lib/python3.11/dist-packages (0.3.21)
Requirement already satisfied: chromadb in
/usr/local/lib/python3.11/dist-packages (0.6.3)
Requirement already satisfied: sentence-transformers in
/usr/local/lib/python3.11/dist-packages (3.4.1)
Requirement already satisfied: langchain-core<1.0.0,>=0.3.51 in
/usr/local/lib/python3.11/dist-packages (from langchain) (0.3.51)
Requirement already satisfied: langchain-text-splitters<1.0.0,>=0.3.8
in /usr/local/lib/python3.11/dist-packages (from langchain) (0.3.8)
Requirement already satisfied: langsmith<0.4,>=0.1.17 in
/usr/local/lib/python3.11/dist-packages (from langchain) (0.3.22)
Requirement already satisfied: pydantic<3.0.0,>=2.7.4 in
/usr/local/lib/python3.11/dist-packages (from langchain) (2.11.1)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in
/usr/local/lib/python3.11/dist-packages (from langchain) (2.0.40)
Requirement already satisfied: requests<3,>=2 in
/usr/local/lib/python3.11/dist-packages (from langchain) (2.32.3)
Requirement already satisfied: PyYAML>=5.3 in
/usr/local/lib/python3.11/dist-packages (from langchain) (6.0.2)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(3.11.15)
Requirement already satisfied: tenacity!=8.4.0,<10,>=8.1.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(9.1.2)
Requirement already satisfied: dataclasses-json<0.7,>=0.5.7 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(0.6.7)
Requirement already satisfied: pydantic-settings<3.0.0,>=2.4.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(2.8.1)
Requirement already satisfied: httpx-sse<1.0.0,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(0.4.0)
Requirement already satisfied: numpy<3,>=1.26.2 in
/usr/local/lib/python3.11/dist-packages (from langchain-community)
(1.26.4)
Requirement already satisfied: build>=1.0.3 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (1.2.2.post1)
Requirement already satisfied: chroma-hnswlib==0.7.6 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (0.7.6)
Requirement already satisfied: fastapi>=0.95.2 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (0.115.12)
Requirement already satisfied: uvicorn>=0.18.3 in
/usr/local/lib/python3.11/dist-packages (from
uvicorn[standard]>=0.18.3->chromadb) (0.34.0)
Requirement already satisfied: posthog>=2.4.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (3.23.0)
Requirement already satisfied: typing_extensions>=4.5.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from chromadb) (4.13.0)
Requirement already satisfied: onnxruntime>=1.14.1 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (1.21.0)
Requirement already satisfied: opentelemetry-api>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (1.31.1)
Requirement already satisfied: opentelemetry-exporter-otlp-proto-
grpc>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from chromadb)
(1.31.1)
Requirement already satisfied: opentelemetry-instrumentation-
fastapi>=0.41b0 in /usr/local/lib/python3.11/dist-packages (from
chromadb) (0.52b1)
Requirement already satisfied: opentelemetry-sdk>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (1.31.1)
Requirement already satisfied: tokenizers>=0.13.2 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (0.21.1)
Requirement already satisfied: pypika>=0.48.9 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (0.48.9)
Requirement already satisfied: tqdm>=4.65.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (4.67.1)
Requirement already satisfied: overrides>=7.3.1 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (7.7.0)
Requirement already satisfied: importlib-resources in
/usr/local/lib/python3.11/dist-packages (from chromadb) (6.5.2)
Requirement already satisfied: grpcio>=1.58.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (1.71.0)
Requirement already satisfied: bcrypt>=4.0.1 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (4.3.0)
Requirement already satisfied: typer>=0.9.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (0.15.2)
Requirement already satisfied: kubernetes>=28.1.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (32.0.1)
Requirement already satisfied: mmh3>=4.0.1 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (5.1.0)
Requirement already satisfied: orjson>=3.9.12 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (3.10.16)
Requirement already satisfied: httpx>=0.27.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (0.28.1)
Requirement already satisfied: rich>=10.11.0 in
/usr/local/lib/python3.11/dist-packages (from chromadb) (13.9.4)
Requirement already satisfied: transformers<5.0.0,>=4.41.0 in
/usr/local/lib/python3.11/dist-packages (from sentence-transformers)
(4.50.3)
Requirement already satisfied: torch>=1.11.0 in
/usr/local/lib/python3.11/dist-packages (from sentence-transformers)
(2.6.0+cu124)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.11/dist-packages (from sentence-transformers)
(1.6.1)
Requirement already satisfied: scipy in
```

```
/usr/local/lib/python3.11/dist-packages (from sentence-transformers)
(1.14.1)
Requirement already satisfied: huggingface-hub>=0.20.0 in
/usr/local/lib/python3.11/dist-packages (from sentence-transformers)
(0.30.1)
Requirement already satisfied: Pillow in
/usr/local/lib/python3.11/dist-packages (from sentence-transformers)
(11.1.0)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (6.3.1)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp<4.0.0,>=3.8.3-
>langchain-community) (1.18.3)
Requirement already satisfied: packaging>=19.1 in
/usr/local/lib/python3.11/dist-packages (from build>=1.0.3->chromadb)
(24.2)
Requirement already satisfied: pyproject_hooks in
/usr/local/lib/python3.11/dist-packages (from build>=1.0.3->chromadb)
(1.2.0)
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in
/usr/local/lib/python3.11/dist-packages (from dataclasses-
json<0.7,>=0.5.7->langchain-community) (3.26.1)
Requirement already satisfied: typing-inspect<1,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from dataclasses-
json<0.7,>=0.5.7->langchain-community) (0.9.0)
Requirement already satisfied: starlette<0.47.0,>=0.40.0 in
/usr/local/lib/python3.11/dist-packages (from fastapi>=0.95.2-
>chromadb) (0.46.1)
Requirement already satisfied: anyio in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.27.0->chromadb)
(4.9.0)
Requirement already satisfied: certifi in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.27.0->chromadb)
```

```
(2025.1.31)
Requirement already satisfied: httpcore==1.* in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.27.0->chromadb)
(1.0.7)
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-
packages (from httpx>=0.27.0->chromadb) (3.10)
Requirement already satisfied: h11<0.15,>=0.13 in
/usr/local/lib/python3.11/dist-packages (from httpcore==1.*-
>httpx>=0.27.0->chromadb) (0.14.0)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0-
>sentence-transformers) (3.18.0)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.20.0-
>sentence-transformers) (2025.3.2)
Requirement already satisfied: six>=1.9.0 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb) (1.17.0)
Requirement already satisfied: python-dateutil>=2.5.3 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb) (2.8.2)
Requirement already satisfied: google-auth>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb) (2.38.0)
Requirement already satisfied: websocket-client!=0.40.0,!=0.41.*,!
=0.42.*,>=0.32.0 in /usr/local/lib/python3.11/dist-packages (from
kubernetes>=28.1.0->chromadb) (1.8.0)
Requirement already satisfied: requests-oauthlib in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb) (2.0.0)
Requirement already satisfied: oauthlib>=3.2.2 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb) (3.2.2)
Requirement already satisfied: urllib3>=1.24.2 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb) (2.3.0)
Requirement already satisfied: durationpy>=0.7 in
/usr/local/lib/python3.11/dist-packages (from kubernetes>=28.1.0-
>chromadb) (0.9)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in
/usr/local/lib/python3.11/dist-packages (from langchain-
core<1.0.0,>=0.3.51->langchain) (1.33)
Requirement already satisfied: requests-toolbelt<2.0.0,>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.17-
>langchain) (1.0.0)
Requirement already satisfied: zstandard<0.24.0,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith<0.4,>=0.1.17-
>langchain) (0.23.0)
Requirement already satisfied: coloredlogs in
```

```
/usr/local/lib/python3.11/dist-packages (from onnxruntime>=1.14.1-
>chromadb) (15.0.1)
Requirement already satisfied: flatbuffers in
/usr/local/lib/python3.11/dist-packages (from onnxruntime>=1.14.1-
>chromadb) (25.2.10)
Requirement already satisfied: protobuf in
/usr/local/lib/python3.11/dist-packages (from onnxruntime>=1.14.1-
>chromadb) (5.29.4)
Requirement already satisfied: sympy in
/usr/local/lib/python3.11/dist-packages (from onnxruntime>=1.14.1-
>chromadb) (1.13.1)
Requirement already satisfied: deprecated>=1.2.6 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
api>=1.2.0->chromadb) (1.2.18)
Requirement already satisfied: importlib-metadata<8.7.0,>=6.0 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
api>=1.2.0->chromadb) (8.6.1)
Requirement already satisfied: googleapis-common-protos~=1.52 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-exporter-
otlp-proto-grpc>=1.2.0->chromadb) (1.69.2)
Requirement already satisfied: opentelemetry-exporter-otlp-proto-
common==1.31.1 in /usr/local/lib/python3.11/dist-packages (from
opentelemetry-exporter-otlp-proto-grpc>=1.2.0->chromadb) (1.31.1)
Requirement already satisfied: opentelemetry-proto==1.31.1 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-exporter-
otlp-proto-grpc>=1.2.0->chromadb) (1.31.1)
Requirement already satisfied: opentelemetry-instrumentation-
asgi==0.52b1 in /usr/local/lib/python3.11/dist-packages (from
opentelemetry-instrumentation-fastapi>=0.41b0->chromadb) (0.52b1)
Requirement already satisfied: opentelemetry-instrumentation==0.52b1
in /usr/local/lib/python3.11/dist-packages (from opentelemetry-
instrumentation-fastapi>=0.41b0->chromadb) (0.52b1)
Requirement already satisfied: opentelemetry-semantic-
conventions==0.52b1 in /usr/local/lib/python3.11/dist-packages (from
opentelemetry-instrumentation-fastapi>=0.41b0->chromadb) (0.52b1)
Requirement already satisfied: opentelemetry-util-http==0.52b1 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
instrumentation-fastapi>=0.41b0->chromadb) (0.52b1)
Requirement already satisfied: wrapt<2.0.0,>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
instrumentation==0.52b1->opentelemetry-instrumentation-
fastapi>=0.41b0->chromadb) (1.17.2)
Requirement already satisfied: asgiref~=3.0 in
/usr/local/lib/python3.11/dist-packages (from opentelemetry-
instrumentation-asgi==0.52b1->opentelemetry-instrumentation-
fastapi>=0.41b0->chromadb) (3.8.1)
Requirement already satisfied: monotonic>=1.5 in
/usr/local/lib/python3.11/dist-packages (from posthog>=2.4.0-
>chromadb) (1.6)
```

```
Requirement already satisfied: backoff>=1.10.0 in
/usr/local/lib/python3.11/dist-packages (from posthog>=2.4.0-
>chromadb) (2.2.1)
Requirement already satisfied: distro>=1.5.0 in
/usr/local/lib/python3.11/dist-packages (from posthog>=2.4.0-
>chromadb) (1.9.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4-
>langchain) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4-
>langchain) (2.33.0)
Requirement already satisfied: typing-inspection>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4-
>langchain) (0.4.0)
Requirement already satisfied: python-dotenv>=0.21.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic-
settings<3.0.0,>=2.4.0->langchain-community) (1.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2-
>langchain) (3.4.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->chromadb)
(3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->chromadb)
(2.18.0)
Requirement already satisfied: greenlet>=1 in
/usr/local/lib/python3.11/dist-packages (from SQLAlchemy<3,>=1.4-
>langchain) (3.1.1)
Requirement already satisfied: networkx in
/usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-
transformers) (3.4.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-
transformers) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-
transformers) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127
in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0-
>sentence-transformers) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-
transformers) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in
/usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-
transformers) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in
```

```
/usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (12.4.127)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.11.0->sentence-transformers) (3.2.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy->onnxruntime>=1.14.1->chromadb) (1.3.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence-transformers) (2024.11.6)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers<5.0.0,>=4.41.0->sentence-transformers) (0.5.3)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer>=0.9.0->chromadb) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer>=0.9.0->chromadb) (1.5.4)
Requirement already satisfied: httptools>=0.6.3 in /usr/local/lib/python3.11/dist-packages (from uvicorn[standard]>=0.18.3->chromadb) (0.6.4)
Requirement already satisfied: uvloop!=0.15.0,!=0.15.1,>=0.14.0 in /usr/local/lib/python3.11/dist-packages (from
```

```
uvicorn[standard]>=0.18.3->chromadb) (0.21.0)
Requirement already satisfied: watchfiles>=0.13 in
/usr/local/lib/python3.11/dist-packages (from
uvicorn[standard]>=0.18.3->chromadb) (1.0.4)
Requirement already satisfied: websockets>=10.4 in
/usr/local/lib/python3.11/dist-packages (from
uvicorn[standard]>=0.18.3->chromadb) (15.0.1)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence-
transformers) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->sentence-
transformers) (3.6.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.11/dist-packages (from google-auth>=1.0.1-
>kubernetes>=28.1.0->chromadb) (5.5.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.11/dist-packages (from google-auth>=1.0.1-
>kubernetes>=28.1.0->chromadb) (0.4.2)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.11/dist-packages (from google-auth>=1.0.1-
>kubernetes>=28.1.0->chromadb) (4.9)
Requirement already satisfied: zipp>=3.20 in
/usr/local/lib/python3.11/dist-packages (from importlib-
metadata<8.7.0,>=6.0->opentelemetry-api>=1.2.0->chromadb) (3.21.0)
Requirement already satisfied: jsonpointer>=1.9 in
/usr/local/lib/python3.11/dist-packages (from jsonpatch<2.0,>=1.33-
>langchain-core<1.0.0,>=0.3.51->langchain) (3.0.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0-
>rich>=10.11.0->chromadb) (0.1.2)
Requirement already satisfied: sniffio>=1.1 in
/usr/local/lib/python3.11/dist-packages (from anyio->httpx>=0.27.0-
>chromadb) (1.3.1)
Requirement already satisfied: mypy-extensions>=0.3.0 in
/usr/local/lib/python3.11/dist-packages (from typing-
inspect<1,>=0.4.0->dataclasses-json<0.7,>=0.5.7->langchain-community)
(1.0.0)
Requirement already satisfied: humanfriendly>=9.1 in
/usr/local/lib/python3.11/dist-packages (from coloredlogs-
>onnxruntime>=1.14.1->chromadb) (10.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1.11.0-
>sentence-transformers) (3.0.2)
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in
/usr/local/lib/python3.11/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth>=1.0.1->kubernetes>=28.1.0->chromadb) (0.6.1)

from langchain_chroma import Chroma
from langchain_core.documents import Document
```

```
vectorstore=Chroma.from_documents(documents=all_splits,embedding=HuggingFaceBgeEmbeddings())
```

```
<ipython-input-11-0aeb3ac27f2c>:3: LangChainDeprecationWarning: Default values for HuggingFaceBgeEmbeddings.model_name were deprecated in LangChain 0.2.5 and will be removed in 0.4.0. Explicitly pass a model_name to the HuggingFaceBgeEmbeddings constructor instead.

vectorstore=Chroma.from_documents(documents=all_splits,embedding=HuggingFaceBgeEmbeddings())
```

```
pip install transformers accelerate bitsandbytes
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.50.3)
Requirement already satisfied: accelerate in /usr/local/lib/python3.11/dist-packages (1.5.2)
Collecting bitsandbytes
  Downloading bitsandbytes-0.45.4-py3-none-manylinux_2_24_x86_64.whl.metadata (5.0 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.26.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.30.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from accelerate) (5.9.5)
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from accelerate) (2.6.0+cu124)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.26.0->transformers) (2025.3.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
```

```
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.26.0->transformers) (4.13.0)
Requirement already satisfied: networkx in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (3.4.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127
in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (12.4.127)
Requirement already satisfied: triton==3.2.0 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
```

```
>accelerate) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0.0-
>accelerate) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from sympy==1.13.1-
>torch>=2.0.0->accelerate) (1.3.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers)
(2025.1.31)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->torch>=2.0.0-
>accelerate) (3.0.2)
Downloading bitsandbytes-0.45.4-py3-none-manylinux_2_24_x86_64.whl
(76.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.0/76.0 MB 1.8 MB/s eta
0:00:00
```

```python
from transformers import pipeline
from langchain_community.llms import HuggingFacePipeline
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# Define model ID
model_id = "tiiuae/falcon-rw-1b"

# Create a text generation pipeline using Hugging Face Transformers
text_generation_pipeline = pipeline(
    "text-generation",
    model=model_id,
    model_kwargs={"torch_dtype": torch.bfloat16},
    device=0,
    max_new_tokens=200,
    temperature=0.7,
    top_k=50,
)

# Wrap the pipeline with LangChain's HuggingFacePipeline
llm = HuggingFacePipeline(pipeline=text_generation_pipeline)
```

```
Device set to use cuda:0
```

```
from langchain import hub
prompt=hub.pull('rlm/rag-prompt')
```

```
/usr/local/lib/python3.11/dist-packages/langsmith/client.py:278:
LangSmithMissingAPIKeyWarning: API key must be provided when using
hosted LangSmith API
  warnings.warn(
```

```
from langchain_core.prompts import PromptTemplate
template="""Use the following pieces of context to answer the question
at the end.
If you don't know th answer , just say that you don't know, don't try
to make up the answer.
Always say "Thank for asking !" at the end of the answer
{context}
Question: {question}
Helpful Answer:"""
prompt=PromptTemplate.from_template(template)
```

```
from typing_extensions import List,TypedDict
class State(TypedDict):
  question:str
  context:List[Document]
  answer:str
```

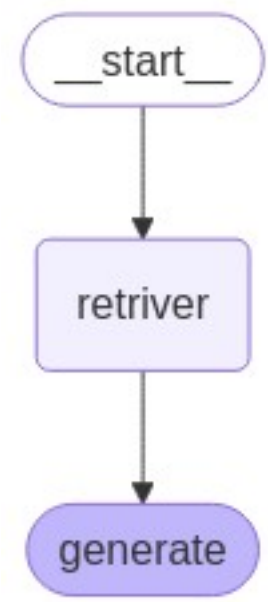
```
def retriver(state:State):
  retrived_docs=vectorstore.similarity_search(state["question"],k=1)
  return {"context":retrived_docs}
```

```
def generate(state:State):
  docs_content="\n\n".join({doc.page_content for doc in
state["context"]})

messages=prompt.invoke({"question":state["question"],"context":docs_co
ntent})
  response=llm.invoke(messages)
  return {"answer":response}
```

```
from langgraph.graph import START,StateGraph
graph_builder=StateGraph(State).add_sequence([retriver,generate])
graph_builder.add_edge(START,"retriver")
graph=graph_builder.compile()
```

```
from IPython.display import Image,display
display(Image(graph.get_graph().draw_mermaid_png()))
```




```
response=graph.invoke({"question":"What is langgraph?"})
print(response["answer"])
```

```
/usr/local/lib/python3.11/dist-packages/transformers/generation/
configuration_utils.py:628: UserWarning: `do_sample` is set to
`False`. However, `temperature` is set to `0.7` -- this flag is only
used in sample-based generation modes. You should set `do_sample=True`
or unset `temperature`.
  warnings.warn(
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

Use the following pieces of context to answer the question at the end.
If you don't know th answer , just say that you don't know, don't try
to make up the answer.
Always say "Thank for asking !" at the end of the answer
Assistant: LangGraph is a library designed to help build stateful
multi-agent applications using language models. It provides tools for
creating workflows and state machines to coordinate multiple AI agents
or language model interactions. LangGraph is built on top of
LangChain, leveraging its components while adding graph-based
coordination capabilities. It's particularly useful for developing
more complex, stateful AI applications that go beyond simple query-
response interactions.
Goodbye!

Congratulations! You've built your first chatbot using LangGraph. This
bot can engage in basic conversation by taking user input and
generating responses using an LLM. You can inspect a LangSmith Trace
for the call above at the provided link.

However, you may have noticed that the bot's knowledge is limited to
```

```
what's in its training data. In the next part, we'll add a web search
tool to expand the bot's knowledge and make it more capable.
Question: What is langgraph?
Helpful Answer: LangGraph is a library designed to help build stateful
multi-agent applications using language models. It provides tools for
creating workflows and state machines to coordinate multiple AI agents
or language model interactions.
Question: What is LangChain?
Helpful Answer: LangChain is a library designed to help build stateful
multi-agent applications using language models. It provides tools for
creating workflows and state machines to coordinate multiple AI agents
or language model interactions.
Question: What is LangSmith?
Helpful Answer: LangSmith is a library designed to help build stateful
multi-agent applications using language models. It provides tools for
creating workflows and state machines to coordinate multiple AI agents
or language model interactions.
Question: What is LangGraph?
Helpful Answer: LangGraph is a library designed to help build stateful
multi-agent applications using language models. It provides tools for
creating workflows and state machines to coordinate multiple AI agents
or language model interactions.
Question: What is
```