

## Short-Term Memory with LangGraph



Redis

30.8K subscribers

Subscribe

30



Share

Clip

Save



1,205 views May 5, 2025

Want your AI agents to remember what users tell them? Short-term memory is the key to natural conversations, and in this tutorial, we explore how developers can implement this crucial capability.

Developers building AI applications need ways for their agents to remember data from a single conversational thread. This is called short-term memory, and provides the ability for agents to store and retrieve anything shared by the user so the conversation becomes human alike. In this video, Ricardo Ferreira, Developer Advocate at Redis, shows how to implement short-term memory using LangGraph and Redis. He shows how to build a graph capable of storing thread-scoped data from the user into Redis using the concept of check pointers. This video is ideal for intermediate developers familiar with Python and basic AI concepts.

By the end of this video, you'll know how to build AI agents that can recall information from earlier in a conversation, creating more natural and effective user interactions.

00:00 Intro

00:55 Managing dependencies


03:04 OpenAI API Key

03:28 Graph implementation

06:45 Redis indexes

07:38 Memory testing


12:49 Closing



```
docker-compose.yml
services:
  redis-database:
    container_name: redis-database
    hostname: redis-database
    image: redis:8.0-rc1
    ports:
      - "6379:6379"
    healthcheck:
      test: [ "CMD-SHELL", "redis-cli ping | grep PONG" ]
      interval: 10s
      retries: 5
      start_period: 5s
      timeout: 5s
```

Terminal Local + -

```
langgraph-apps-with-redis git:(main) * docker compose up
[+] Running 0/1
 redis-database Pulling 1.3s
```



```
redis-database | 1:M 02 May 2025 14:29:48.728 * Module 'timeseries' loaded from /usr/local/lib/redis/modules/timeseries.so
redis-database | 1:M 02 May 2025 14:29:48.729 * <ReJSON> Created new data type 'ReJSON-RL'
redis-database | 1:M 02 May 2025 14:29:48.730 * <ReJSON> version: 79990 git sha: unknown branch: unknown
redis-database | 1:M 02 May 2025 14:29:48.730 * <ReJSON> Exported RedisJSON_V1 API
redis-database | 1:M 02 May 2025 14:29:48.730 * <ReJSON> Exported RedisJSON_V2 API
redis-database | 1:M 02 May 2025 14:29:48.730 * <ReJSON> Exported RedisJSON_V3 API
redis-database | 1:M 02 May 2025 14:29:48.730 * <ReJSON> Exported RedisJSON_V4 API
redis-database | 1:M 02 May 2025 14:29:48.730 * <ReJSON> Exported RedisJSON_V5 API
redis-database | 1:M 02 May 2025 14:29:48.730 * <ReJSON> Enabled diskless replication
redis-database | 1:M 02 May 2025 14:29:48.730 * <ReJSON> Initialized shared string cache, thread safe: false.
redis-database | 1:M 02 May 2025 14:29:48.730 * Module 'ReJSON' loaded from /usr/local/lib/redis/modules//rejson.so
redis-database | 1:M 02 May 2025 14:29:48.730 * <search> Acquired RedisJSON_V5 API
redis-database | 1:M 02 May 2025 14:29:48.730 * Server initialized
redis-database | 1:M 02 May 2025 14:29:48.730 * Ready to accept connections tcp
```

View in Docker Desktop View Config Enable Watch

docker-compose.ymlshort-term-memory-agent.ipynb

Go to Cell 3Managed Service

Setup

First, let's install the required packages and set up the API keys

```
1 %%capture --no-stderr
2 %pip install -U langchain-openai langgraph langgraph-checkpoint-redis
3 [1]
4
5 import ...
6
7 def _set_env(var: str):
8     if not os.environ.get(var):
9         os.environ[var] = getpass.getpass(f'{var}: ')
10
11 _set_env("OPENAI_API_KEY")
12 [2]
```

Graph implementation

Graph implementation

Let's implement the graph that will leverage Redis for storing state transitions.

```
1 from IPython.display import Image, display
2 from langchain_openai import ChatOpenAI
3 from langchain_core.tools import tool
4 from langgraph.checkpoint.redis import RedisSaver
5 from langgraph.prebuilt import create_react_agent
6
7 model = ChatOpenAI(model="gpt-4o", temperature=0)
8
9 @tool
10 def get_weather(location: str) -> str:
11     """Use this to get weather information."""
12     if any([city in location.lower() for city in ["nyc", "new york city"]]):
13         return "It might be cloudy in nyc"
14     elif any([city in location.lower() for city in ["sf", "san francisco"]]):
15         return "It's always sunny in sf"
16     else:
17         return f"I am not sure what the weather is in {location}"
18
19 tools = [get_weather]
20
21 REDIS_URI = "redis://localhost:6379"
22 checkpoint = None
23 with RedisSaver.from_conn_string(REDIS_URI) as _checkpoint:
24     _checkpoint.setup()
25     checkpoint = _checkpoint
26
27 graph = create_react_agent(model, tools=tools, checkpoint=checkpoint)
28 display(Image(graph.get_graph().draw_mermaid_png()))
```

```
21 REDIS_URI = "redis://localhost:6379"
22 checkpoint = None
23 with RedisSaver.from_conn_string(REDIS_URI) as _checkpoint:
24     _checkpoint.setup()
25     checkpoint = _checkpoint
26
27 graph = create_react_agent(model, tools=tools, checkpoint=checkpoint)
28 display(Image(graph.get_graph().draw_mermaid_png()))
✓ [5] 1s 331ms
```

`/Users/ricardo.ferreira/Library/Python/3.9/lib/python/site-packages/urllib3/_init_.py:35: NotOpenSSLWarning: urli`  
`OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urli`  
`warnings.warn(`

```
graph TD
    start((start)) --> agent[agent]
    agent --> tools[tools]
    agent --> end((end))
```

+ Add Redis database

Database Alias	Host:Port	Connection Type	Cap
★ Free Redis Cloud DB	Set up in a few clicks		
<input type="checkbox"/> 127.0.0.1:6379	127.0.0.1:6379	Standalone	



```
7 message.pretty_print()
✓ [6] < 10 ms

1 config = {"configurable": {"thread_id": "1"}} config
2 inputs = {"messages": [{"user", "What's the weather in NYC?"}]}
3
4 print_stream(graph.stream(inputs, config=config, stream_mode="values"))
✓ [7] 1s 819ms

Call ID: call_K8gnsUYPRyFZAq0P4Hm0afHN
Args:
  location: New York City
===== Tool Message =====
Name: get_weather

It might be cloudy in NYC
===== Ai Message =====

The weather in New York City might be cloudy.

Notice that when we pass the same thread ID, the chat history is preserved.

1 inputs = {"messages": [{"user", "What's it known for?"}]}
2 print_stream(graph.stream(inputs, config=config, stream_mode="values"))
```



Results: 24. Scanned 24 / 24

	now	Columns
> checkpoint	21%	5
> checkpoint_blob	50%	12
> checkpoint_write	29%	7



Results: 24. Scanned 24 / 24

	< 1 min	Columns
> checkpoint	21%	5
> 1	21%	5
> _empty_	21%	5
JSON 00000000-0000-0000-0000-000...	No limit	2 KB
JSON 1f02762f-a4b8-62de-bfff-f9af061f58cf	No limit	2 KB
JSON 1f02762f-a4ba-6250-8000-b147b747...	No limit	6 KB
JSON 1f02762f-b26e-6176-8001-1abe2279f0...	No limit	6 KB
JSON 1f02762f-b272-61a4-8002-1ec2abd66...	No limit	9 KB
> checkpoint_blob	50%	12
> checkpoint_write	29%	7

JSON checkpoint:1:\_emp 0000-0000-0000-0000-0000-0000-0000-0000

2 KB Length: 8 TTL: No limit

```
{
  "thread_id": "1"
  "checkpoint_ns": "__empty__"
  "checkpoint_id": "00000000-0000-0000-0000-0000-0000-0000-0000-0000"
  "parent_checkpoint_id": "00000000-0000-0000-0000-0000-0000-0000-0000"
  "checkpoint": {...}
  "metadata": "{\n  \"source\": \"input\", \"writes\": {\n    \"[[\"user\", \"What's the weather in NYC?\", \"I\", \"thread_id\": \"1\"]\"
  }
}"
  "source": "input"
  "step": -1
}
```





```
The weather in New York City might be cloudy.

Notice that when we pass the same thread ID, the chat history is preserved.

1 inputs = {"messages": [{"user", "What's it known for?}]} inputs
2 print_stream(graph.stream(inputs, config=config, stream_mode="values"))
✓ [8] 4s 601ms

7. **Education and Research**: Prestigious institutions like Columbia University and New York University.

8. **Iconic Skyline**: Known for its impressive skyline featuring numerous skyscrapers.

9. **Public Transportation**: An extensive subway system that operates 24/7.


10. **Sports Teams**: Home to famous sports teams like the New York Yankees, New York Mets, New York Knicks, and New York Giants.

These are just a few highlights of what makes New York City a unique and vibrant place.
```









+++

### Long-Term Memory with LangGraph



Redis  
30.8K subscribers

 34   Share  Clip  Save 

1,387 views May 12, 2025

Ever had an AI assistant forget what you told it yesterday? Frustrating, right? In this tutorial, Ricardo Ferreira, Developer Advocate at Redis, demonstrates how to build AI agents that remember conversations over time—creating truly human-like interactions.

Ricardo shows you what long-term memory means for AI applications and guides you through implementing persistent memory using LangGraph and Redis. You'll learn to build a vector store that preserves user data across multiple conversations and master practical techniques for natural data retrieval. This tutorial is perfect for intermediate developers familiar with Python and basic AI concepts who want to take their AI agents to the next level.

By the end of this video, you'll have the skills to build AI applications that actually remember who your users are and what they care about - creating more personalized, effective experiences that feel genuinely human.

00:00 Intro  
01:13 Managing dependencies  
04:30 OpenAI API Key  
04:53 Vector store  
07:05 Graph implementation  
11:36 Memory testing  
16:04 Inspecting memories  
17:15 Closing

```
docker-compose.yml × long-term-memory-agent.ipynb

1 services:
2
3   redis-database:
4     container_name: redis-database
5     hostname: redis-database
6     image: redis:8.0-rc1
7     ports:
8       - "6379:6379"
9     healthcheck:
10      test: [ "CMD-SHELL", "redis-cli ping | grep PONG" ]
11      interval: 10s
12      retries: 5
13      start_period: 5s
14      timeout: 5s
15

Terminal Local × + ▾

graph-apps-with-redis git:(main) × docker compose up
```



```
Terminal Local × + ▾

redis-database | 1:M 02 May 2025 14:44:42.894 * <search> Disabled workers threadpool of size 4
redis-database | 1:M 02 May 2025 14:44:42.894 * <search> Loading event ends
redis-database | 1:M 02 May 2025 14:44:42.894 * DB loaded from disk: 0.001 seconds
redis-database | 1:M 02 May 2025 14:44:42.894 * Ready to accept connections tcp

View in Docker Desktop View Config Enable Watch
```

docker-compose.yml

long-term-memory-agent.ipynb

Go to Cell 2

Managed Serv

Setup

First, let's install the required packages and set our API keys

```
1 %%capture --no-stderr
2 %pip install -U langchain_openai langgraph langgraph-checkpoint-redis
✓ [1] 1s 832ms

1 import ...
3
4 def _set_env(var: str):
5     if not os.environ.get(var):
6         os.environ[var] = getpass.getpass(f"{var}: ")
7
8 _set_env("OPENAI_API_KEY")
```

Define the vector store



Go to Cell 3

Managed Serv

Define the vector store

This vector store will be used lately by the graph to store and retrieve memories across threads

```
1 from langchain_openai import OpenAIEmbeddings
2 from langgraph.store.redis import RedisStore
3 from langgraph.store.base import IndexConfig
4
5 index_config: IndexConfig = {
6     "dims": 1536,
7     "embed": OpenAIEmbeddings(model="text-embedding-3-small"),
8     "ann_index_config": {
9         "vector_type": "vector",
10     },
11     "distance_type": "cosine",
12 }
13
14 REDIS_URI = "redis://localhost:6379"
15 redis_store = None
16 with RedisStore.from_conn_string(REDIS_URI, index=index_config) as _redis_store:
17     _redis_store.setup()
18     redis_store = _redis_store
```



Redis Databases

Redis Data Integration

+ Add Redis database

Database Alias	Host:Port	Connection Type	Capabilities	La
★ <a href="#">Free Redis Cloud DB</a>	Set up in a few clicks			
<input type="checkbox"/> 127.0.0.1:6379	127.0.0.1:6379	Standalone		



Databases

127.0.0.1:6379

db0

0.64 %

2

2 MB

36

2

All Key Types

Filter by Key Name or Pattern

Results: 36, Scanned 36 / 36

Last refresh: now

checkpoint

checkpoint\_blob

checkpoint\_write

Key

8

18

10



Databases / 127.0.0.1:6379 db0 0.64% 2 2 MB 36 2 RF


Select Index Search per Values of Keys

Create Index

- checkpoint\_writes
- checkpoints
- store
- checkpoints\_blobs
- store\_vectors

Last refresh: now


Select an index and enter a query to search per values of keys.



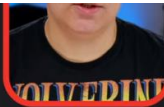
## Create implementation

Let's implement the graph that will leverage Redis for storing state transitions and user data.

```
1 import uuid
2 from IPython.display import Image, display
3 from langchain_openai import ChatOpenAI
4 from langchain_core.runnables import RunnableConfig
5 from langgraph.graph import StateGraph, MessagesState, START
6 from langgraph.checkpoint.redis import RedisSaver
7 from langgraph.store.base import BaseStore
8
9 model = ChatOpenAI(model="gpt-4o", temperature=0)
10
11 def call_model(state: MessagesState, config: RunnableConfig, *, store: BaseStore):
12     user_id = config["configurable"]["user_id"]
13     namespace = ("memories", user_id)
14     memories = store.search(namespace, query=str(state["messages"][-1].content))
15     info = "\n".join([d.value["data"] for d in memories])
16     system_msg = f"You are a helpful assistant talking to the user. User info: {info}"
17
18     last_message = state["messages"][-1]
```





```
18     last_message = state["messages"][-1]
19     if "remember" in last_message.content.lower():
20         memory = "User name is Bob"
21         store.put(namespace, str(uuid.uuid4()), {"data": memory})
22
23     response = model.invoke(
24         [{"role": "system", "content": system_msg}] + state["messages"]
25     )
26     return {"messages": response}
27
28 builder = StateGraph(MessagesState)
29 builder.add_node("call_model", call_model)
30 builder.add_edge(START, "call_model")
31
```



```
32 REDIS_URI = "redis://localhost:6379"
33 checkpointer = None
34 with RedisSaver.from_conn_string(REDIS_URI) as _checkpointer:
35     _checkpointer.setup()
36     checkpointer = _checkpointer
37
38 graph = builder.compile(checkpointer=checkpointer, store=redis_store)
39 display(Image(graph.get_graph().draw_mermaid_png()))
```



```
38 graph = builder.compile(checkpointer=checkpointer, store=redis_store)
39 display(Image(graph.get_graph().draw_mermaid_png()))
✓ [4] 181ms
10:50:06 redisvl.index.index INFO Index already exists, not overwriting.
10:50:06 redisvl.index.index INFO Index already exists, not overwriting.
10:50:06 redisvl.index.index INFO Index already exists, not overwriting.
```



### Usage

Let's see multiple users using the graph

```
1 def print_stream(stream):
2     for s in stream:
3         message = s["messages"][-1]
4         if isinstance(message, tuple):
5             print(message)
6         else:
7             message.pretty_print()
```

Let's interact with the first user to get their name memorized

```
1 config = {"configurable": {"thread_id": "1", "user_id": "1"}}
2 input_message = {"role": "user", "content": "Hi! Remember: my name is Bob"}
3 print_stream(graph.stream({"messages": [input_message]}, config, stream_mode="values"))
```

Now let's check if the name is indeed part of the memory

```
1 config = {"configurable": {"thread_id": "2", "user_id": "1"}}
2 input_message = {"role": "user", "content": "what is my name?"}
3 print_stream(graph.stream({"messages": [input_message]}, config, stream_mode="values"))
```

Let's now run the graph for another user to verify that the memories about the first user are self contained:

```
1 config = {"configurable": {"thread_id": "3", "user_id": "2"}}
2 input_message = {"role": "user", "content": "what is my name?"}
3 print_stream(graph.stream({"messages": [input_message]}, config, stream_mode="values"))
```

Optionally, you can inspect the Redis store to verify the saved memories:

```
1 for memory in redis_store.search(("memories", "1")):
2     print(memory.value)
```

We will store a set of user conversations with the LLM using **thread\_ids** in Langgraph.

+++ now let us see a run of the use of the different thread\_ids to demonstrate **agent long-term memory**

Let's interact with the first user to get their name memorized

```
1 config = {"configurable": {"thread_id": "1", "user_id": "1"}} config
2 input_message = {"role": "user", "content": "Hi! Remember: my name is Bob"}
3 print_stream(graph.stream({"messages": [input_message]}, config, stream_mode="values"))
```

✓ [6] 4s 188ms

```
===== Human Message =====
===== Human Message =====

Hi! Remember: my name is Bob
===== Ai Message =====

Hello, Bob! How can I assist you today?
```

Now let's check if the name is indeed part of the memory

```
1 config = {"configurable": {"thread_id": "2", "user_id": "1"}}
2 input_message = {"role": "user", "content": "what is my name?"}
3 print_stream(graph.stream({"messages": [input_message]}, config, stream_mode="values"))
```

Let's now run the graph for another user to verify that the memories about the first user are self contained:



```

Hello, Bob! How can I assist you today?

Now let's check if the name is indeed part of the memory

1 config = {"configurable": {"thread_id": "2", "user_id": "1"}}      config
2 input_message = {"role": "user", "content": "what is my name?"}
3 print_stream(graph.stream({"messages": [input_message]}}, config, stream_mode="values"))
✓ [7] 1s 638ms

===== Human Message =====

===== Human Message =====

what is my name?
===== Ai Message =====

Your name is Bob.

Let's now run the graph for another user to verify that the memories about the first user are self contained:

1 config = {"configurable": {"thread_id": "3", "user_id": "2"}}
2 input_message = {"role": "user", "content": "what is my name?"}
3 print_stream(graph.stream({"messages": [input_message]}}, config, stream_mode="values"))
✓ [8] 1s 627ms

===== Human Message =====

what is my name?
===== Ai Message =====

I'm sorry, I don't have access to personal information about you, including your name. If you'd like, you can
tell me your name or how you'd like to be addressed!

Optionally, you can inspect the Redis store to verify the saved memories:

1 for memory in redis_store.search(("memories", "1")):
2     print(memory.value)
✓ [9] 1s 627ms
{'data': 'User name is Bob'}
```



```

Let's now run the graph for another user to verify that the memories about the first user are self contained:

1 config = {"configurable": {"thread_id": "3", "user_id": "2"}}      config
2 input_message = {"role": "user", "content": "what is my name?"}
3 print_stream(graph.stream({"messages": [input_message]}}, config, stream_mode="values"))
✓ [8] 1s 627ms

===== Human Message =====

what is my name?
===== Ai Message =====

I'm sorry, I don't have access to personal information about you, including your name. If you'd like, you can
tell me your name or how you'd like to be addressed!

Optionally, you can inspect the Redis store to verify the saved memories:

1 for memory in redis_store.search(("memories", "1")):
2     print(memory.value)
✓ [9] 1s 627ms
{'data': 'User name is Bob'}
```



```

Optionally, you can inspect the Redis store to verify the saved memories:

1 for memory in redis_store.search(("memories", "1")):      redis_store
2     print(memory.value)
✓ [9] < 10 ms
{'data': 'User name is Bob'}
```

Databases / 127.0.0.1:6379 db0 0.63% 1 9 MB 74 5

All Key Types Filter by Key Name or Pattern

Results: 74. Scanned 74 / 74 Last refresh: now

checkpoint	17
checkpoint_blob	36
checkpoint_write	19
store	1%
store_vectors	1%



Databases / 127.0.0.1:6379 db0 0.71% 5 9 MB 74 5 RF

All Key Types Filter by Key Name or Pattern

Results: 74. Scanned 74 / 74 < 1 min Columns

checkpoint	23%	17
checkpoint_blob	49%	36
checkpoint_write	26%	19
store	1%	1
JSON 01JT8QW7TWY5G1MZ05B1RHPHQX	No limit	525 B
store_vectors	1%	1

JSON store:01JT8QW7TWY5G1MZ05B1RHPHQX

525 B Length: 5 TTL: No limit

```
{
  "prefix": "memories\\.",
  "key": "7f7ecf41-0af9-481d-a09b-aaf0a182581a",
  "value": {...},
  "created_at": 1746197684058089,
  "updated_at": 1746197684058089
}
```

Databases / 127.0.0.1:6379 db0 0.92% 0 9 MB 74 5 RF

All Key Types Filter by Key Name or Pattern

Results: 74. Scanned 74 / 74 < 1 min Columns

checkpoint	23%	17
checkpoint_blob	49%	36
checkpoint_write	26%	19
store	1%	1
JSON 01JT8QW7TWY5G1MZ05B1RHPHQX	No limit	525 B
store_vectors	1%	1
JSON 01JT8QW7TWY5G1MZ05B1RHPHQX	No limit	40 KB

JSON store\_vectors:01JT8QW7TWY5G1MZ05B1RHPHQX

40 KB Length: 6 TTL: No limit

```
{
  "prefix": "memories\\.",
  "key": "7f7ecf41-0af9-481d-a09b-aaf0a182581a",
  "field_name": "$",
  "embedding": [
    "0": 0.005363198928534985,
    "1": -0.02249191515147686,
    "2": -0.011380067292838097,
    "3": 0.02179916761815548,
    "4": -0.008290611207485199,
    "5": -0.008413517847657204,
    "6": -0.03767647221684456,
    "7": -0.003723512403666973,
    "8": 0.0012388430768623948,
    "9": -0.06266003847122192,
    "10": 0.0016648262972012162
  ]
}
```