How to cache LLM calls

Contents

This notebook covers how to cache results of individual LLM calls.

```
from langchain.llms import OpenAI
```

In Memory Cache

```
import langchain
from langchain.cache import InMemoryCache
langchain.llm_cache = InMemoryCache()
```

```
# To make the caching really obvious, lets use a slower model.
llm = OpenAI(model_name="text-davinci-002", n=2, best_of=2)
```

```
%%time
# The first time, it is not yet in cache, so it should take longer
llm("Tell me a joke")
```

```
CPU times: user 30.7 ms, sys: 18.6 ms, total: 49.3 ms
Wall time: 791 ms
```

Skip to main content

```
"\n\nWhy couldn't the bicycle stand up by itself? Because it was...two tired!"
```

```
%%time
# The second time it is, so it goes faster
llm("Tell me a joke")
```

```
CPU times: user 80 µs, sys: 0 ns, total: 80 µs
Wall time: 83.9 µs
```

```
"\n\nWhy couldn't the bicycle stand up by itself? Because it was...two tired!"
```

SQLite Cache

```
!rm .langchain.db
```

```
# We can do the same thing with a SQLite cache
from langchain.cache import SQLiteCache
langchain.llm_cache = SQLiteCache(database_path=".langchain.db")
```

```
%%time
# The first time, it is not yet in cache, so it should take longer
llm("Tell me a joke")
```

```
CPU times: user 17 ms, sys: 9.76 ms, total: 26.7 ms
Wall time: 825 ms
```

```
'\n\nWhy did the chicken cross the road?\n\nTo get to the other side.'
```

```
%%time
# The second time it is, so it goes faster
llm("Tell me a joke")
```

Skip to main content

```
CPU times: user 2.46 ms, sys: 1.23 ms, total: 3.7 ms
Wall time: 2.67 ms
```

```
'\n\nWhy did the chicken cross the road?\n\nTo get to the other side.'
```

Redis Cache

```
# We can do the same thing with a Redis cache
# (make sure your local Redis instance is running first before running this
example)
from redis import Redis
from langchain.cache import RedisCache
langchain.llm_cache = RedisCache(redis_=Redis())
```

```
%%time
# The first time, it is not yet in cache, so it should take longer
llm("Tell me a joke")
```

```
%%time
# The second time it is, so it goes faster
llm("Tell me a joke")
```

SQLAlchemy Cache

```
# You can use SQLAlchemyCache to cache with any SQL database supported by
SQLAlchemy.

# from langchain.cache import SQLAlchemyCache
# from sqlalchemy import create_engine

# engine = create_engine("postgresql://postgres:postgres@localhost:5432/postgres")
# langchain.llm_cache = SQLAlchemyCache(engine)
```

Custom SQLAlchemy Schemas

Skip to main content

```
# You can define your own declarative SQLAlchemyCache child class to customize the
schema used for caching. For example, to support high-speed fulltext prompt
indexing with Postgres, use:

from sqlalchemy import Column, Integer, String, Computed, Index, Sequence
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy_utils import TSVectorType
from langchain.cache import SQLAlchemyCache

Base = declarative_base()


class FulltextLLMCache(Base):  # type: ignore
    """Postgres table for fulltext-indexed LLM Cache"""

    __tablename__ = "llm_cache_fulltext"
    id = Column(Integer, Sequence('cache_id'), primary_key=True)
    prompt = Column(String, nullable=False)
    llm = Column(String, nullable=False)
    idx = Column(Integer)
    response = Column(String)
    prompt_tsv = Column(TSVectorType(), Computed("to_tsvector('english', llm || '
' || prompt)", persisted=True))
    __table_args__ = (
        Index("idx_fulltext_prompt_tsv", prompt_tsv, postgresql_using="gin"),
    )

engine = create_engine("postgresql://postgres:postgres@localhost:5432/postgres")
langchain.llm_cache = SQLAlchemyCache(engine, FulltextLLMCache)
```

Optional Caching

You can also turn off caching for specific LLMs should you choose. In the example below, even though global caching is enabled, we turn it off for a specific LLM

```
llm = OpenAI(model_name="text-davinci-002", n=2, best_of=2, cache=False)
```

```
%%time
llm("Tell me a joke")
```

```
CPU times: user 5.8 ms, sys: 2.71 ms, total: 8.51 ms
Wall time: 745 ms
```

Skip to main content

```
'\n\nWhy did the chicken cross the road?\n\nTo get to the other side!'
```

```
%%time
llm("Tell me a joke")
```

```
CPU times: user 4.91 ms, sys: 2.64 ms, total: 7.55 ms
Wall time: 623 ms
```

```
'\n\nTwo guys stole a calendar. They got six months each.'
```

Optional Caching in Chains

You can also turn off caching for particular nodes in chains. Note that because of certain interfaces, its often easier to construct the chain first, and then edit the LLM afterwards.

As an example, we will load a summarizer map-reduce chain. We will cache results for the map-step, but then not freeze it for the combine step.

```
llm = OpenAI(model_name="text-davinci-002")
no_cache_llm = OpenAI(model_name="text-davinci-002", cache=False)
```

```
from langchain.text_splitter import CharacterTextSplitter
from langchain.chains.mapreduce import MapReduceChain

text_splitter = CharacterTextSplitter()
```

```
with open('../../../state_of_the_union.txt') as f:
    state_of_the_union = f.read()
texts = text_splitter.split_text(state_of_the_union)
```

```
from langchain.docstore.document import Document
docs = [Document(page_content=t) for t in texts[:3]]
from langchain.chains.summarize import load_summarize_chain
```

Skip to main content

```
chain = load_summarize_chain(llm, chain_type="map_reduce", reduce_llm=no_cache_llm)
```

```
%%time
chain.run(docs)
```

```
CPU times: user 452 ms, sys: 60.3 ms, total: 512 ms
Wall time: 5.09 s
```

```
'\n\nPresident Biden is discussing the American Rescue Plan and the Bipartisan
Infrastructure Law, which will create jobs and help Americans. He also talks about
his vision for America, which includes investing in education and infrastructure.
In response to Russian aggression in Ukraine, the United States is joining with
European allies to impose sanctions and isolate Russia. American forces are being
mobilized to protect NATO countries in the event that Putin decides to keep moving
west. The Ukrainians are bravely fighting back, but the next few weeks will be
hard for them. Putin will pay a high price for his actions in the long run.
Americans should not be alarmed, as the United States is taking action to protect
its interests and allies.'
```

When we run it again, we see that it runs substantially faster but the final answer is different. This is due to caching at the map steps, but not at the reduce step.

```
%%time
chain.run(docs)
```

```
CPU times: user 11.5 ms, sys: 4.33 ms, total: 15.8 ms
Wall time: 1.04 s
```

```
'\n\nPresident Biden is discussing the American Rescue Plan and the Bipartisan
Infrastructure Law, which will create jobs and help Americans. He also talks about
his vision for America, which includes investing in education and infrastructure.'
```