

# Async API

LangChain provides async support for Chains by leveraging the [asyncio](#) library.

Async methods are currently supported in `LLMChain` (through `arun`, `apredict`, `acall`) and `LLMMathChain` (through `arun` and `acall`), `ChatVectorDBChain`, and [QA chains](#). Async support for other chains is on the roadmap.

```
import asyncio
import time

from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

def generate_serially():
    llm = OpenAI(temperature=0.9)
    prompt = PromptTemplate(
        input_variables=["product"],
        template="What is a good name for a company that makes {product}?",
    )
    chain = LLMChain(llm=llm, prompt=prompt)
    for _ in range(5):
        resp = chain.run(product="toothpaste")
        print(resp)
```

```
async def async_generate(chain):
    resp = await chain.arun(product="toothpaste")
    print(resp)

async def generate_concurrently():
    llm = OpenAI(temperature=0.9)
    prompt = PromptTemplate(
        input_variables=["product"],
        template="What is a good name for a company that makes {product}?",
    )
    chain = LLMChain(llm=llm, prompt=prompt)
    tasks = [async_generate(chain) for _ in range(5)]
    await asyncio.gather(*tasks)

s = time.perf_counter()
# If running this outside of Jupyter, use asyncio.run(generate_concurrently())
await generate_concurrently()
elapsed = time.perf_counter() - s
print("\033[1m" + f"Concurrent executed in {elapsed:0.2f} seconds." + "\033[0m")

s = time.perf_counter()
generate_serially()
elapsed = time.perf_counter() - s
print("\033[1m" + f"Serial executed in {elapsed:0.2f} seconds." + "\033[0m")
```

BrightSmile Toothpaste Company

BrightSmile Toothpaste Co.

BrightSmile Toothpaste

Gleaming Smile Inc.

SparkleSmile Toothpaste  
Concurrent executed in 1.54 seconds.

BrightSmile Toothpaste Co.

MintyFresh Toothpaste Co.

SparkleSmile Toothpaste.

Pearly Whites Toothpaste Co.

BrightSmile Toothpaste.  
Serial executed in 6.38 seconds.