CS 696 Applied Large Language Models
Spring Semester, 2025
Doc 21 News, LangChain 2
Apr 15, 2025

News - Skin Cancer Screening

https://www.bbc.com/news/articles/czd3ygd7mrno

UK NHS using AI to screen for skin cancer

Used in 20+ hospitals

Detected over 14,000 cases of Cancer

DolphinGemma

Gemma model trained on Dolphin sounds

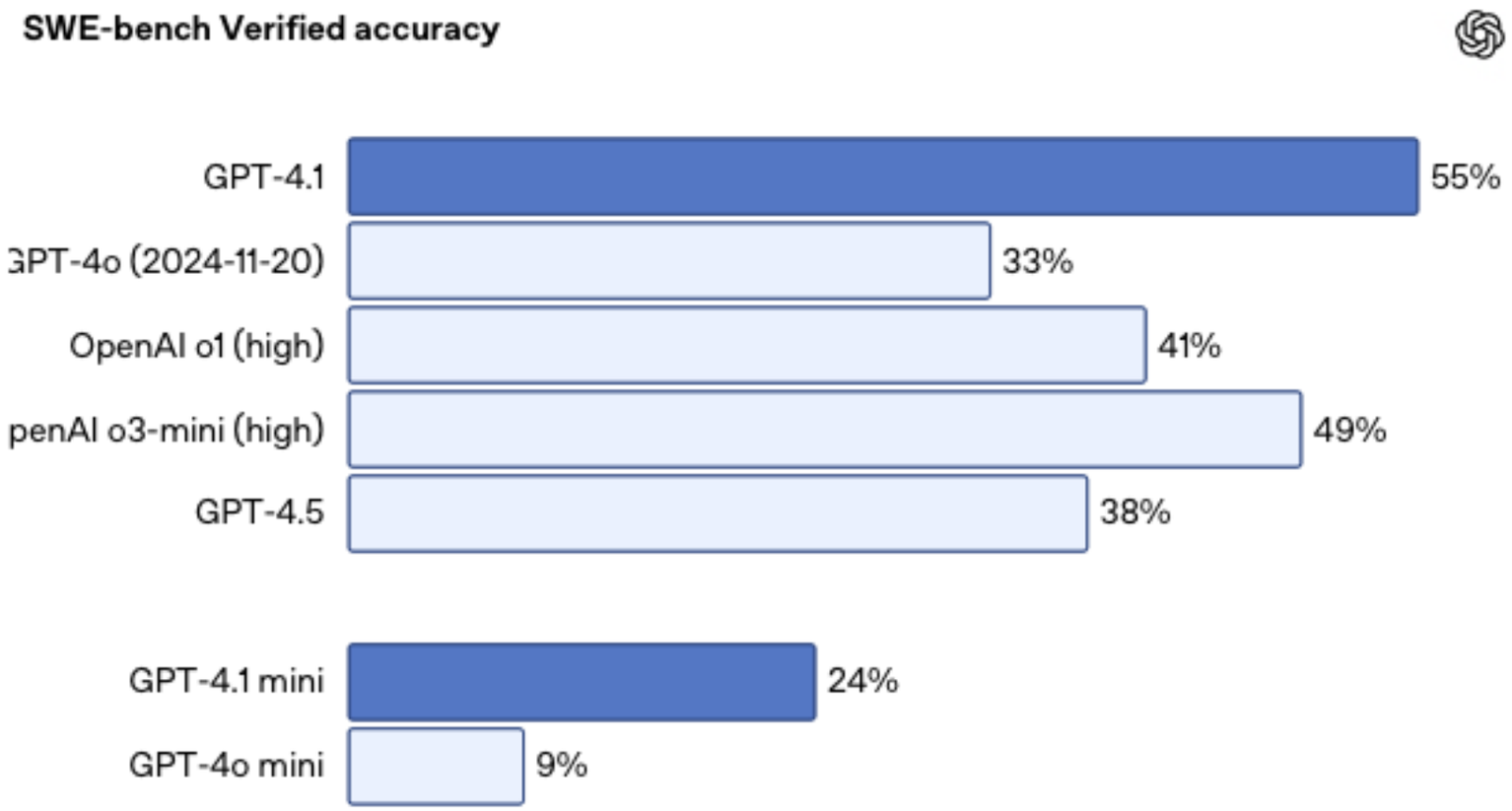
~400 parameter model

Runs on Pixel phone

GPT-4.1 (no chat)

API access only

1 million tokens of context



Price Per 1M tokens

Model (Prices are per 1M tokens)	Input	Cached input	Output	Blended Pricing*
gpt-4.1	$2.00	$0.50	$8.00	$1.84
gpt-4.1-mini	$0.40	$0.10	$1.60	$0.42
gpt-4.1-nano	$0.10	$0.025	$0.40	$0.12

Batch API Rates

50% of the standard rate

Rag Apps - How to Run

As a function call in an application

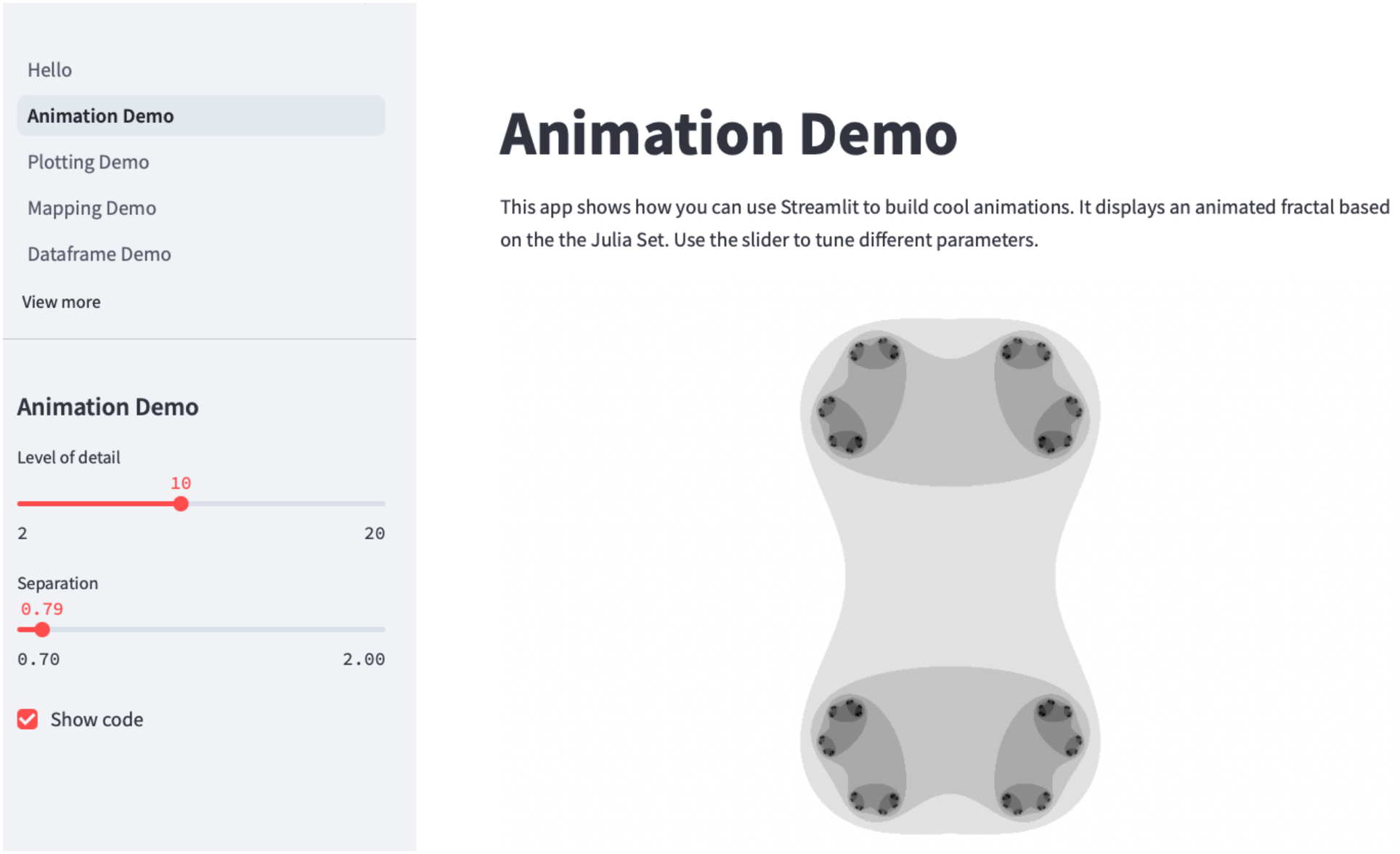
UK Cancer Screening

As a standalone app with an interface

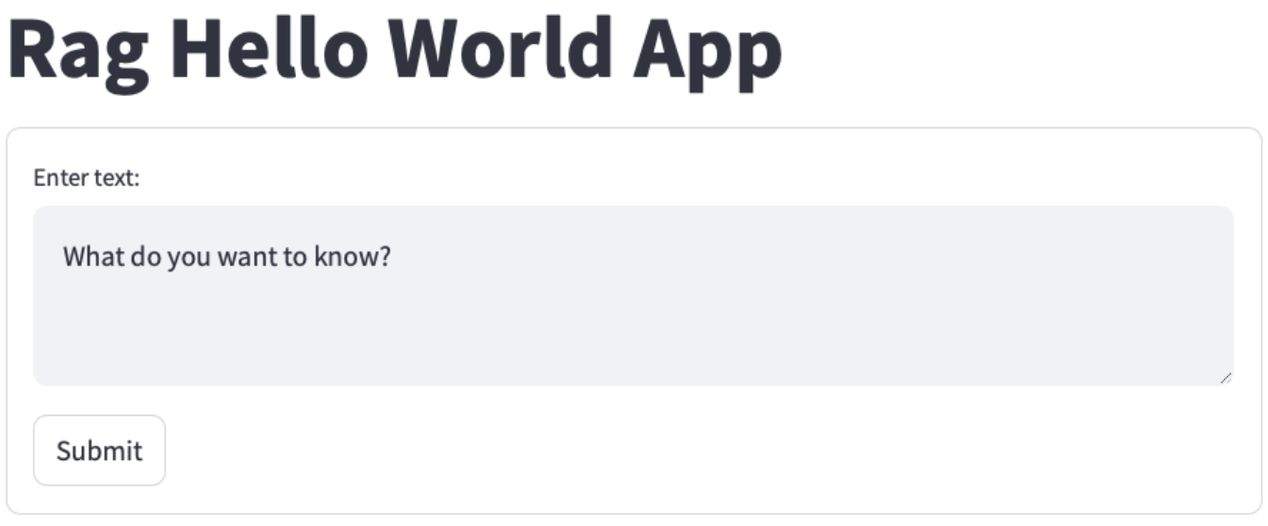
Streamlit

https://streamlit.io/

Library/framework for developing Webpages for Data apps



Barebones Chat App




```
import streamlit as st
from langchain_openai import ChatOpenAI

openai_api_key = “xxx"
llm = ChatOpenAI(model=“gpt-4.1-nano",openai_api_key=openai_api_key)

st.title("Rag Hello World App")

def generate_response(input_text):
    response = llm(input_text)
    st.info(response.content)

with st.form("my_form"):
    text = st.text_area("Enter text:", "What do you want to know?")
    submitted = st.form_submit_button("Submit")
    if submitted:
        generate_response(text)
```

LangChain, LangSmith, LangGraph

Standard interface for large language models and related technologies

LangChain
- Chat Models
- Semantic Search
- Classification
- Extraction

LangGraph
- Assemble LangChain components into apps

LangSmith
- Trace, Monitor & evaluate LLM app

LangChain Expression Language (LCEL)

Chain together different components

|

Like unix pipe

A common functional programming construct

ls *.py | wc -l

cat file.txt | tr -s ' ' '\n' | sort | uniq -c | sort -nr | head -n 1

LCEL Example

```
from langchain_core.prompts import ChatPromptTemplate
from langchain.chat_models import init_chat_model

model = init_chat_model("gpt-4o-mini", model_provider="openai")
system_template = "Translate the following from English into {language}"

prompt_template = ChatPromptTemplate.from_messages(
    [("system", system_template), ("user", "{text}")]
)

prompt = prompt_template.invoke({"language": "Italian", "text": "hi!"})
response = model.invoke(prompt)

chain = prompt_template | model
response = chain.invoke({"language": "Italian", "text": "hi!"})
```

Standard Prompt

```
from langchain_openai.chat_models import ChatOpenAI
from langchain_core.prompts import (SystemMessagePromptTemplate,
ChatPromptTemplate)

template = """
You are a creative consultant brainstorming names for businesses.

You must follow the following principles:
{principles}

Please generate a numerical list of five catchy names for a start-up in the
{industry} industry that deals with {context}?

Here is an example of the format:
1. Name1
2. Name2
3. Name3
4. Name4
5. Name5
"""
```

LangChain Code

```
model = ChatOpenAI()
system_prompt = SystemMessagePromptTemplate.from_template(template)
chat_prompt = ChatPromptTemplate.from_messages([system_prompt])

chain = chat_prompt | model

result = chain.invoke({
    "industry": "medical",
    "context":'"creating AI solutions by automatically summarizing patient
    records"',
    "principles":'"1. Each name should be short and easy to
    remember. 2. Each name should be easy to pronounce.
    3. Each name should be unique and not already taken by another company."'
})

print(result.content)
```

Semantic Search Engine

Documents and document loaders
Text splitters
Embeddings
Vector stores and retrievers

Loaders - PDF

```
from langchain_community.document_loaders import PyPDFLoader

file_path = "SeedLM.pdf"
loader = PyPDFLoader(file_path)
docs = loader.load()

print(len(docs), " Pages")
print(f"{docs[0].page_content[:200]}\n")
print(docs[0].metadata)
```

```
13  Pages
arXiv:2410.10714v2  [cs.LG]  16 Oct 2024
SeedLM: Compressing LLM W eights into Seeds of
Pseudo-Random Generators
Rasoul Shafipour 1, David Harrison 1, Maxwell Horton 1, Jeffrey Marker 1, Houman Bedayat

{'producer': 'GPL Ghostscript 10.01.2', 'creator': 'LaTeX with hyperref', 'creationdate':
'2024-10-16T20:19:23-04:00', 'moddate': '2024-10-16T20:19:23-04:00', 'title': '', 'subject': '',
'author': '', 'keywords': '', 'source': 'SeedLM.pdf', 'total_pages': 13, 'page': 0, 'page_label': '1'}
```

Document loaders

Webpages: 8 Different Loaders

PDFs: 12

Cloud Providers: 15

Social Platforms: 2

Messaging Services: 5

Common File Loaders: 6

- Unstructored Loader knows 59 different file types

RecursiveCharacterTextSplitter

A document may be too coarse - break into pieces

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
file_path = "SeedLM.pdf"
loader = PyPDFLoader(file_path)

docs = loader.load()

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=50, chunk_overlap=5, add_start_index=True, strip_whitespace=True
)
all_splits = text_splitter.split_documents(docs)

print(len(all_splits), "Splits")
print(f"{all_splits[13].page_content[:50]}\n")
print(f"{all_splits[14].page_content[:50]}\n")
print(all_splits[13].metadata)
```

```
print(len(all_splits), "Splits")
print(f"{all_splits[13].page_content[:50]}\n")
print(f"{all_splits[14].page_content[:50]}\n")
print(all_splits[13].metadata)
```

```
1269 Splits
compression met hod that uses seeds of pseudo-

random generators to encode and compress model

{'producer': 'GPL Ghostscript 10.01.2', 'creator': 'LaTeX with hyperref', 'creationdate':
'2024-10-16T20:19:23-04:00', 'moddate': '2024-10-16T20:19:23-04:00', 'title': '', 'subject': '', 'author': '',
'keywords': '', 'source': 'SeedLM.pdf', 'total_pages': 13, 'page': 0, 'page_label': '1', 'start_index': 530}
```

RecursiveCharacterTextSplitter

Argument	Type	Description
chunk_size	int	The maximum number of characters in each chunk.
chunk_overlap	int	Number of characters that overlap between chunks. Helps maintain context continuity.
separators	List[str]	A list of separators to recursively try when splitting the text. Defaults to ["\n\n", "\n", " ", ""].
length_function	Callable	A function to measure the "length" of a chunk. Defaults to Python's built-in len. Can be customized (e.g., to count tokens using tiktoken).
is_separator_regex	bool	If True, treats the separators list as regex patterns. Defaults to False.

Embedings

```
from langchain_openai import OpenAIEmbeddings

embeddings = OpenAIEmbeddings(model="text-embedding-3-large")

vector_1 = embeddings.embed_query(all_splits[0].page_content)
vector_2 = embeddings.embed_query(all_splits[1].page_content)

assert len(vector_1) == len(vector_2)
print(f"Generated vectors of length {len(vector_1)}\n")
print(vector_1[:10])
```

```
Generated vectors of length 3072

[-0.013346947729587555, 0.009683598764240742, -0.019879184663295746,
0.011466722935438156, 0.06292132288217545, 0.021079702302813 53,
-0.009436433203518 39, 0.071360267698764 8, -0.002829164499416 9474,
0.026658585295081 14]
```

Embeddings - Why

D1 In a recent survey we discovered that most students want to take Machine Learning

D2 Along the coast is a popular course to sail to Alaska

Q1 What is a popular CS course?

Embedding Vector Cosine Similarity

D1 0.68

D2 0.42

Q2 What class do Computer Science students want to take?

Embedding Vector Cosine Similarity

D1 0.46

D2 0.15

InMemoryVectorStore

```
from langchain_core.vectorstores import InMemoryVectorStore

vector_store = InMemoryVectorStore(embeddings)

ids = vector_store.add_documents(documents=all_splits)
ids
```

```
['14652af7-6707-4fd0-bf41-935bd3e69e53',
 '0e2bb8df-adb3-4cdb-b2c6-890c0083828b',
 '9d88aebf-0cb9-43a7-bee5-21198288f5ad',
 'e0fde7e0-120d-4a8b-bd10-2d5ff45a7dfe',
 '6a82b456-716c-4e46-8855-6c9f4a01e372',
 '75e13870-9838-42d0-a2b4-9cded0b9747b',
 'c44714df-e3bc-4474-b6a7-69b608799152',
```

Search

```
results = vector_store.similarity_search(
    "What is Linear Feedback Shift Register"
)
print(results[0])
```

page_content='3.1 Linear Feedback Shift Register (LFSR)
A Linear Feedback Shift Register (LFSR) is a simple yet effective type of shift register, ideal for generating

pseudo-random binary sequences. The primary advantages of LFSRs in hardware include cost-effectiveness and

minimal resource consumption due to their straightforward implementation with basic flip-flops and XOR gates.

This simplicity facilitates rapid and efficient sequence ge neration, which is integral to our compression technique.

An LFSR operation can be characterized by its length K (which determines the number of bits in its shift register)

and its feedback polynomial. T o generate next pseudo-rando m number in the sequence, each bit in the register is

```
results = vector_store.similarity_search_with_score("What is Linear Feedback Shift Registe
for k in range(0, len(results)):
    doc, score = results[k]
    print(f"Score: {score}\n")
```

```
Score: 0.6775136765485695

Score: 0.5617602220667822

Score: 0.4898027003828273 6

Score: 0.4380443250025306 5
```

Why In-Memory Database

L1 Cache REference	0.5 ns
L2 cache Reference	5 ns
Main Memory reference	100 ns
Read 4k randomly from SSD	150,000 ns
Read 1 MB sequentially from memory	250,000 ns
Read 1 MB sequentially from SSD	1,000,000 ns
Read 1 MB sequentially from disk	20,000,00 ns

https://gist.github.com/jboner/2841832

Vector Databases

Feature	FAISS	Pinecone	Weaviate	Milvus	Qdrant	Chroma	Elastic/ OpenSearch	Redis
Open Source	✅	❌	✅	✅	✅	✅	✅	✅
Cloud Managed Option	❌	✅	✅	✅ (Zilliz)	✅	❌	✅	✅
ANN Search	✅	✅	✅	✅	✅	✅	✅ (limited)	✅
Metadata Filtering	❌	✅	✅	✅	✅	✅	✅	✅
Hybrid Search	❌	✅	✅	✅	✅	❌	✅	✅
GPU Support	✅	❌	❌	✅	❌	❌	❌	❌

Facebook AI Similarity Search (FAISS)

Table 1

Feature	Description
Vector Similarity Search	Fast search for high-dimensional vectors
Distance Metrics	Inner Product, (Cosine) L2, custom
Index Types	Flat, IVF, PQ, HNSW
GPU Support	CUDA acceleration for search
Quantization	Reduce memory usage
Clustering	Built-in KMeans support
Persistence	Save/load indexes
Batching	Batch search supported
Language Support	Python & C++ APIs

LangChain FAISS vs Full FAISS Api

LangChain has a wrapper for full FAISS implmentations

- Higher level of abstraction
- Adds metadata filtering
- Adds some hybrid search
- Better for RAG

LangChain API

https://python.langchain.com/api_reference/community/vectorstores/langchain_community.vectorstores.faiss.FAISS.html#langchain_community.vectorstores.faiss.FAISS.load_local

Full API

https://github.com/facebookresearch/faiss/wiki/Installing-Faiss

Facebook AI Similarity Search (FAISS)

Our Data

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
file_path = "SeedLM.pdf"
loader = PyPDFLoader(file_path)

docs = loader.load()

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000, chunk_overlap=200, add_start_index=True, strip_whitespace=True
)
all_splits = text_splitter.split_documents(docs)
```

Database

```
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS

db = FAISS.from_documents(all_splits, OpenAIEmbeddings())
```

Search

```
query = "What is Linear Feedback Shift Register"
docs = db.similarity_search(query)
print(docs[0])
```

page_content='3.1 Linear Feedback Shift Register (LFSR)
A Linear Feedback Shift Register (LFSR) is a simple yet effective type of shift register, ideal for generating pseudo-random binary sequences. The primary advantages of LFSRs in hardware include cost-effectiveness and minimal resource consumption due to their straightforward implementation with basic flip-flops and XOR gates. This simplicity facilitates rapid and efficient sequence ge neration, which is integral to our compression technique. An LFSR operation can be characterized by its length K (which determines the number of bits in its shift register) and its feedback polynomial. T o generate next pseudo-rando m number in the sequence, each bit in the register is first shifted to the next position. Then, the new bit entering the register is calculated as a linear combination of certain bits of the current state as specified by the feedback polynomial, typically implemented by XOR operations.'

metadata={'producer': 'GPL Ghostscript 10.01.2', 'creator': 'LaTeX with hyperref', 'creationdate': '2024-10-16T20:19:23-04:00', 'moddate': '2024-10-16T20:19:23-04:00', 'title': '', 'subject': '', 'author': '', 'keywords': '',
'source': 'SeedLM.pdf', 'total_pages': 13,
'page': 3,
‘**page_label': '4',** 'start_index': 0}

Asynchronous Operations

docs = await db.**a**similarity_search(query)

Embedding Search

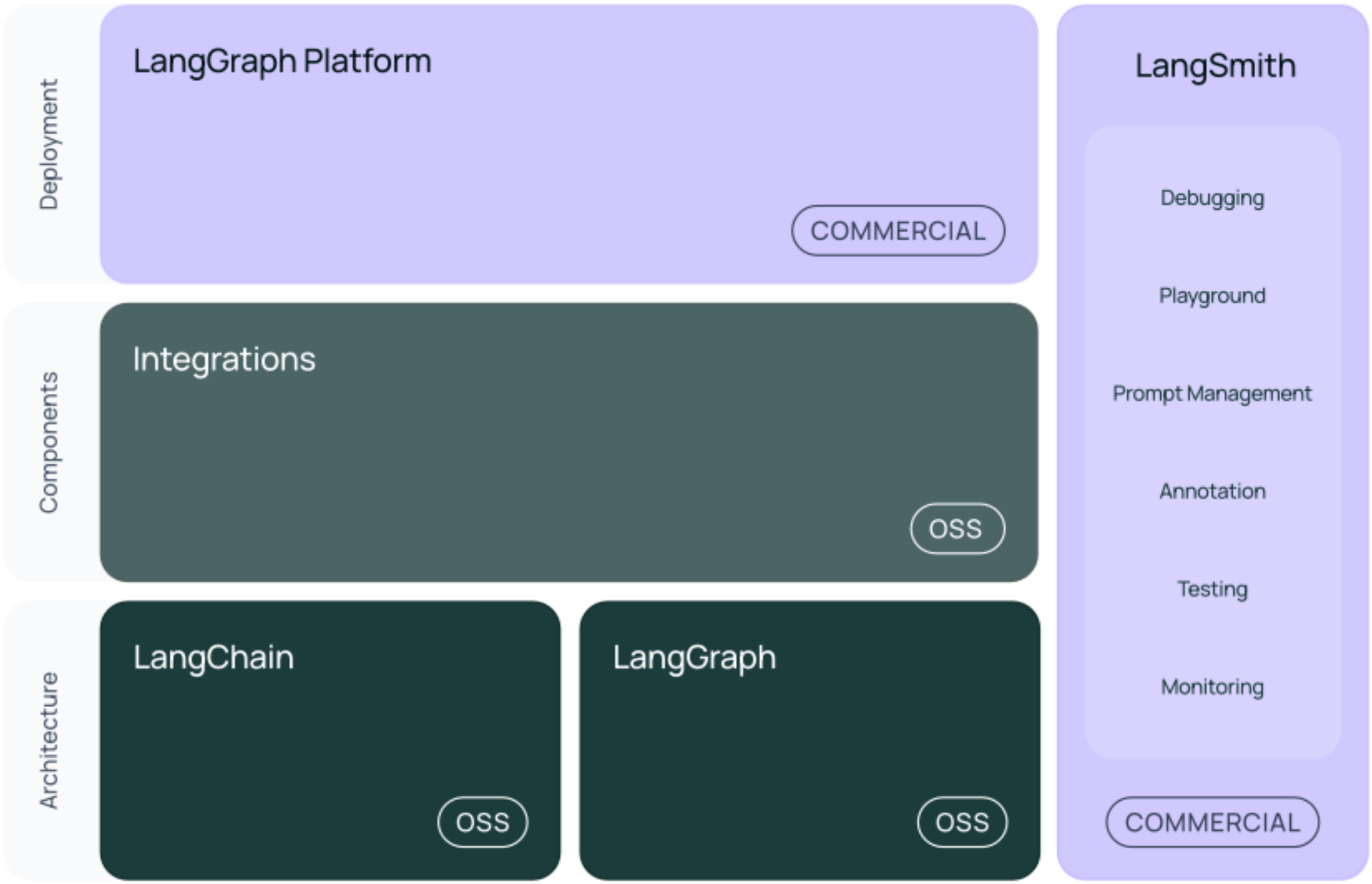
```
embedding_vector = OpenAIEmbeddings().embed_query(query)
docs = db.similarity_search_by_vector(embedding_vector)
```

Saving and Loading

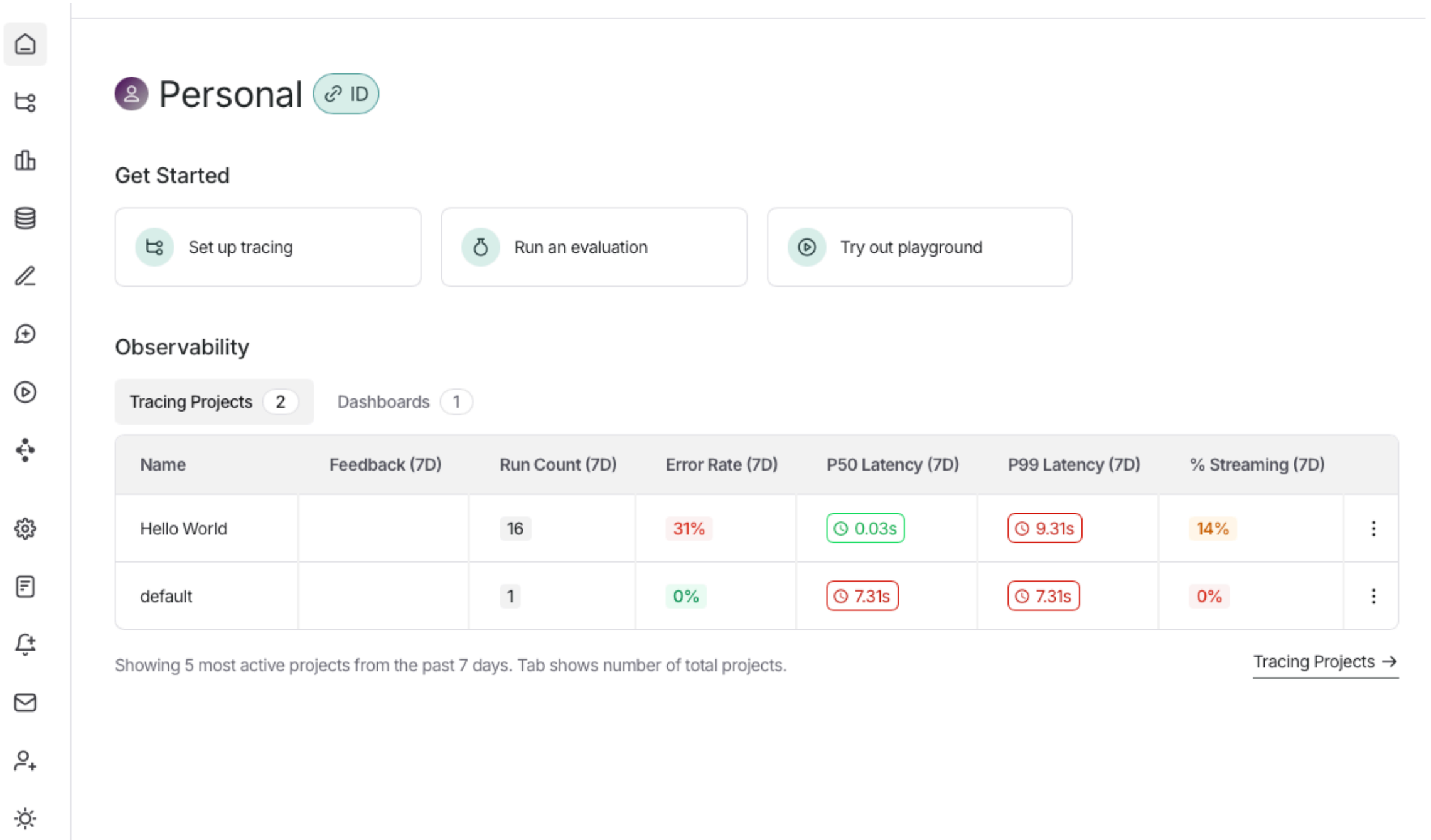
```
db.save_local("llmDB")
```

```
recoverdDB = FAISS.load_local(“llmDB",OpenAIEmbeddings(),
                              allow_dangerous_deserialization=True)
```

LangSmith


Deployment
LangGraph Platform
COMMERCIAL
Components
Integrations
OSS
Architecture
LangChain
OSS
LangGraph
OSS
LangSmith
Debugging
Playground
Prompt Management
Annotation
Testing
Monitoring
COMMERCIAL

LangSmith - Monitoring



Runs Threads Monitor Setup

1 filter | Last 7 days | Root Runs | LLM Calls | All Runs | Columns

Name	Input	Output	Error	Start Time	Latency	Dataset
ChatOpenAI	human: How many Rs...	ai: ```json { "title":...		4/6/2025, 8:23:56 ...	10.10s	
ChatPromptTemplate	How many Rs are in s...	{"messages":[{"c...		4/6/2025, 8:23:56 ...	0.00s	
ChatPromptTemplate	How many Rs are in s...		KeyError('Input to...	4/6/2025, 8:16:48 PM	0.00s	
ChatOpenAI	human: How many Rs...	ai: ```json { "title":...		4/6/2025, 8:05:47 ...	4.10s	
ChatPromptTemplate	How many Rs are in s...	{"messages":[{"c...		4/6/2025, 8:05:47 ...	0.00s	
ChatOpenAI	human: How many Rs...	ai: ```json { "title":...		4/6/2025, 8:05:14 PM	4.59s	
ChatPromptTemplate	How many Rs are in s...	{"messages":[{"c...		4/6/2025, 8:05:14 PM	0.00s	
ChatPromptTemplate	How many Rs are in s...		KeyError('Input to...	4/6/2025, 8:04:36 ...	0.00s	

TRACE

Waterfall | Show All

ChatPromptTemplate

0.00s

ChatPromptTemplate

ID

Run | Feedback | Metadata

Error

Copy

KeyError('Input to ChatPromptTemplate is missing variables {\'\\n "title"\'}. Expected: [\'\\n "title"\', \'foo\'] Received: [\'foo\']\nNote: if you intended {\n "title"} to be part of the string and not a variable, please escape it with double curly braces like: \'{{\n "title"}}\'.\nFor troubleshooting, visit: https://python.langchain.com/docs/troubleshooting/errors/INVALID_PROMPT_INPUT ')Traceback (most recent call last):

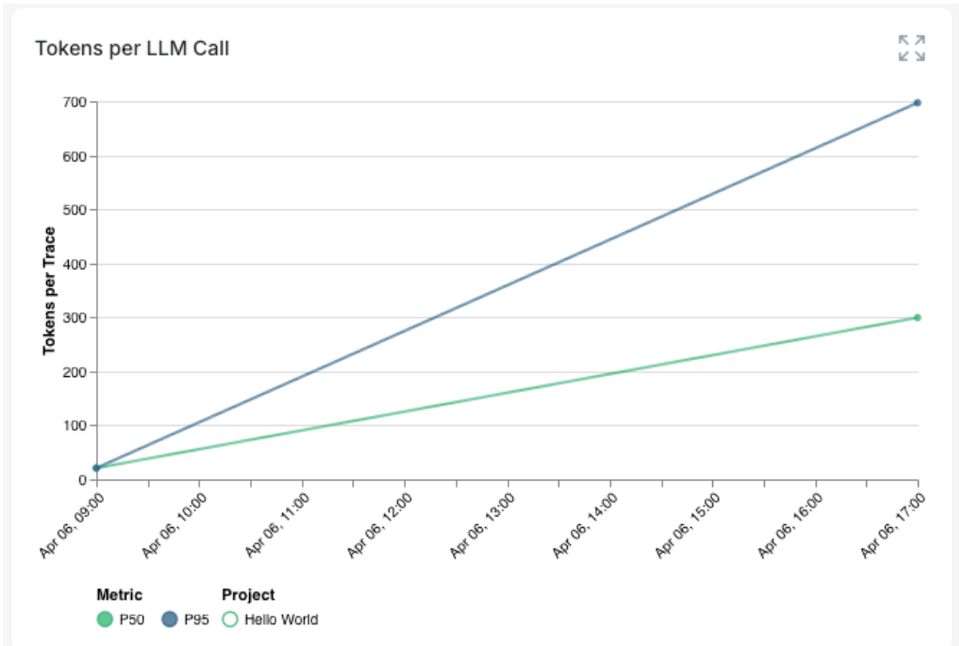
File "/Users/rwhitney/anaconda3/lib/python3.12/site-packages/langchain_core/runnables/base.py", line 1933, in _call_with_config
context.run(

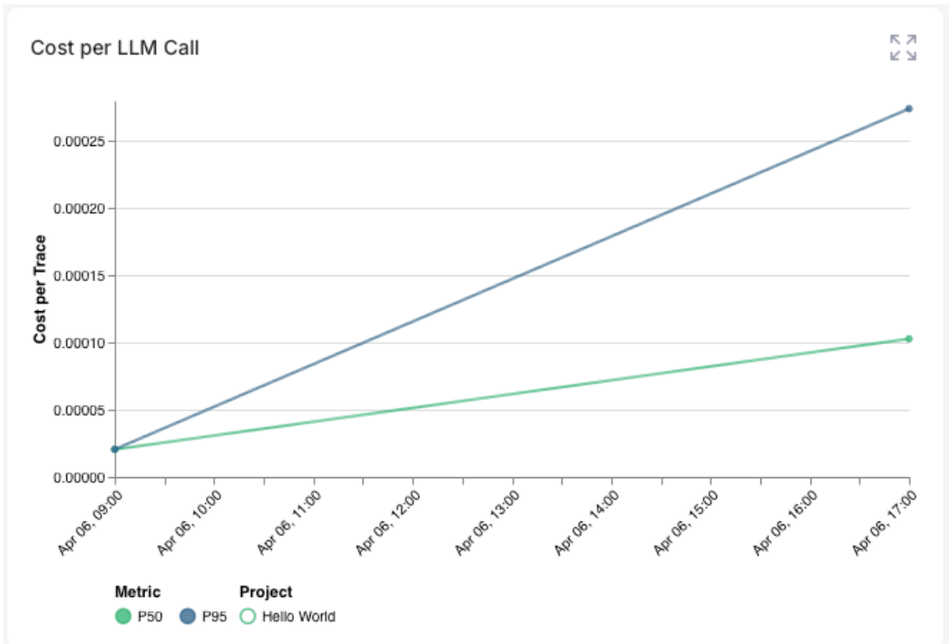
File "/Users/rwhitney/anaconda3/lib/python3.12/site-packages/langchain_core/runnables/config.py", line 428, in call_func_with_variable_args
return func(input, **kwargs) # type: ignore[call-arg]
^^^^^^^^^^^^^^^^^^^^^

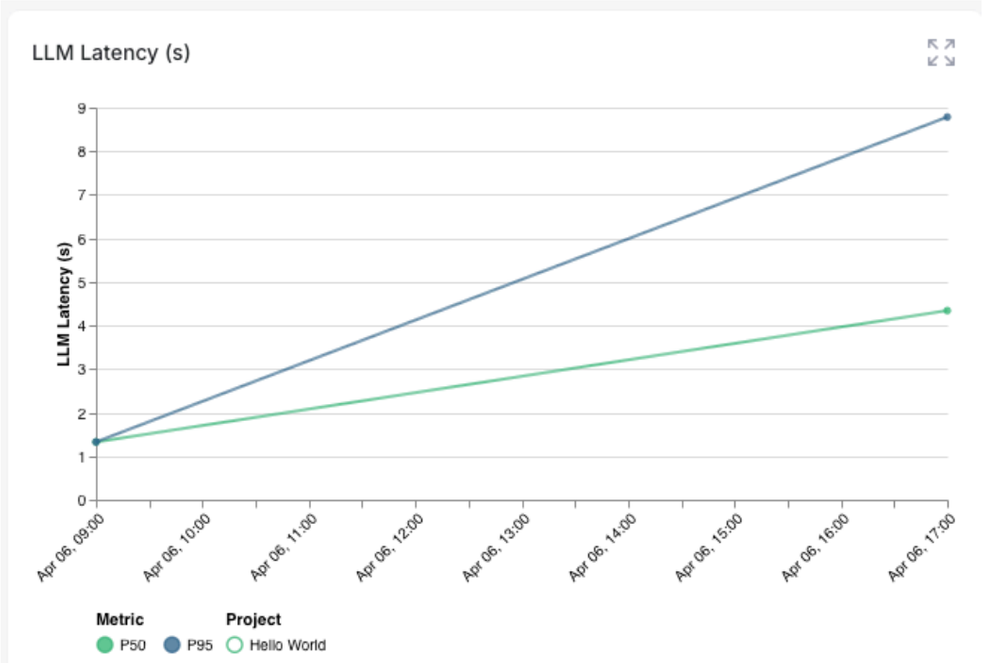
STATUS

Error

Monitoring


Tokens per LLM Call
Tokens per Trace
700
600
500
400
300
200
100
0
Apr 06, 09:00
Apr 06, 10:00
Apr 06, 11:00
Apr 06, 12:00
Apr 06, 13:00
Apr 06, 14:00
Apr 06, 15:00
Apr 06, 16:00
Apr 06, 17:00
Metric
Project
P50
P95
Hello World


Cost per LLM Call
Cost per Trace
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Apr 06, 09:00
Apr 06, 10:00
Apr 06, 11:00
Apr 06, 12:00
Apr 06, 13:00
Apr 06, 14:00
Apr 06, 15:00
Apr 06, 16:00
Apr 06, 17:00
Metric
Project
P50
P95
Hello World


LLM Latency (s)
LLM Latency (s)
9
8
7
6
5
4
3
2
1
0
Apr 06, 09:00
Apr 06, 10:00
Apr 06, 11:00
Apr 06, 12:00
Apr 06, 13:00
Apr 06, 14:00
Apr 06, 15:00
Apr 06, 16:00
Apr 06, 17:00
Metric
Project
P50
P95
Hello World

LangGraph

State machine/workflow engine for LLM apps

Graph of Behaviors

- Nodes - steps
- Edges - logic/routing between steps

Control Over Multi-Step Logic

Support for Loops and Branches

Built-in State Management

- Global state

Debuggable and Observable

First Example

```
from langgraph.graph import START, StateGraph

def add_node(state, config):
    return {"x": state["x"] + 1}

builder = StateGraph(dict)
builder.add_node(add_node)  # node name will be 'my_node'
builder.add_edge(START, "add_node")
graph = builder.compile()
graph.invoke({"x": 1})
```

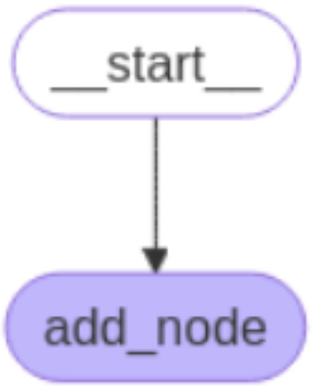
```
 {'x': 2}
```

Show the Graph

```
from langgraph.graph import START, StateGraph

def add_node(state, config):
    return {"x": state["x"] + 1}

builder = StateGraph(dict)
builder.add_node(add_node)  # node name will be 'my_node'
builder.add_edge(START, "add_node")
graph = builder.compile()
graph
```



Multiple Requests

```
from langgraph.graph import START, StateGraph

def add_node(state, config):
    return {"x": state["x"] + 1}

builder = StateGraph(dict)
builder.add_node(add_node)  # node name will be 'my_node'
builder.add_edge(START, "add_node")
graph = builder.compile()
result1 = graph.invoke({"x": 1})
result2 = graph.invoke({"x": 10})
print(result1)
print(result2)
```

```
{'x': 2}
{'x': 11}
```

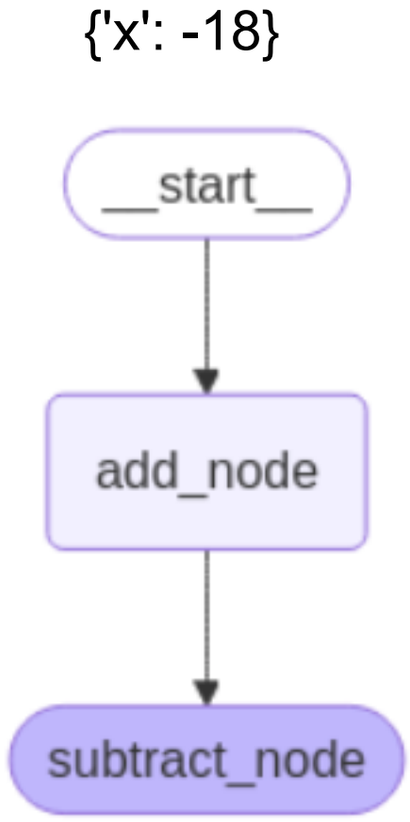
Multiple Nodes

```
from langgraph.graph import START, StateGraph

def add_node(state, config):
    return {"x": state["x"] + 1}

def subtract_node(state, config):
    return {"x": state["x"] - 20}

builder = StateGraph(dict)
builder.add_node(add_node)  # node name will be 'my_node'
builder.add_node(subtract_node)
builder.add_edge(START, "add_node")
builder.add_edge("add_node", "subtract_node")
graph = builder.compile()
result1 = graph.invoke({"x": 1})
print(result1)
graph
```



Stream Example

```
from typing import Annotated
from langchain_openai import ChatOpenAI
from typing_extensions import TypedDict

from langgraph.graph import StateGraph
from langgraph.graph.message import add_messages

class State(TypedDict):
    messages: Annotated[list, add_messages]

graph_builder = StateGraph(State)

llm = ChatOpenAI(model="gpt-4o-mini")

def chatbot(state: State):
    return {"messages": [llm.invoke(state["messages"])]}

graph_builder.add_node("chatbot", chatbot)
graph_builder.set_entry_point("chatbot")
graph_builder.set_finish_point("chatbot")
graph = graph_builder.compile()
```

```
def stream_graph_updates(user_input: str):
    for event in graph.stream({"messages": [{"role": "user", "content": user_input}]}):
        for value in event.values():
            print("Assistant:", value["messages"][-1].content)

while True:
    try:
        user_input = input("User: ")
        if user_input.lower() in ["quit", "exit", "q"]:
            print("Goodbye!")
            break
        stream_graph_updates(user_input)
    except:
        # fallback if input() is not available
        user_input = "What do you know about LangGraph?"
        print("User: " + user_input)
        stream_graph_updates(user_input)
        break
```

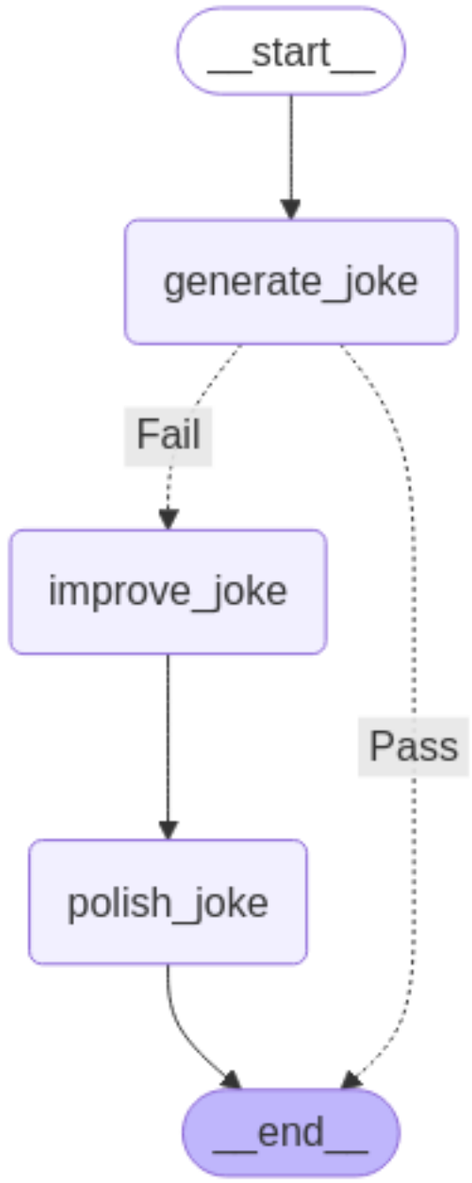
Interaction

```
User:  Hello
Assistant: Hello! How can I assist you today?
User:  What is 1 + 1
Assistant: 1 + 1 equals 2.
User:  How many R's are in Strawberry
Assistant: The word "strawberry" contains three "R's."
User:  ↑↓ for history. Search history with c-↑/c·
```

Joke Example

If first request is not good enough

Have two rounds of improvement



State

```
from typing_extensions import TypedDict
from langgraph.graph import StateGraph, START, END
from IPython.display import Image, display


# Graph state
class State(TypedDict):
    topic: str
    joke: str
    improved_joke: str
    final_joke: str
```

Nodes

```
def generate_joke(state: State):
    """First LLM call to generate initial joke"""
   msg = llm.invoke(f"Write a short joke about {state['topic']}")
    return {"joke": msg.content}

def check_punchline(state: State):
    """Gate function to check if the joke has a punchline""
    # Simple check - does the joke contain "?" or "!"
    if "?" in state["joke"] or "!" in state["joke"]:
        return "Fail"
    return "Pass"

def improve_joke(state: State):
    """Second LLM call to improve the joke"""
    msg = llm.invoke(f"Make this joke funnier by adding wordplay: {state['joke']}")
    return {"improved_joke": msg.content}

def polish_joke(state: State):
    """Third LLM call for final polish"""
    msg = llm.invoke(f"Add a surprising twist to this joke: {state['improved_joke']}")
    return {"final_joke": msg.content}
```

Constructing the Graph

```
workflow = StateGraph(State)

# Add nodes
workflow.add_node("generate_joke", generate_joke)
workflow.add_node("improve_joke", improve_joke)
workflow.add_node("polish_joke", polish_joke)

# Add edges to connect nodes
workflow.add_edge(START, "generate_joke")
workflow.add_conditional_edges(
    "generate_joke", check_punchline, {"Fail": "improve_joke", "Pass": END}
)
workflow.add_edge("improve_joke", "polish_joke")
workflow.add_edge("polish_joke", END)

# Compile
chain = workflow.compile()
```

Running

```
state = chain.invoke({"topic": "cats"})
print("Initial joke:")
print(state["joke"])
print("\n--- --- ---\n")
if "improved_joke" in state:
    print("Improved joke:")
    print(state["improved_joke"])
    print("\n--- --- ---\n")
    print("Final joke:")
    print(state["final_joke"])
else:
    print("Joke failed quality gate - no punchline detected!")
```

Output

Initial joke:
Why was the cat sitting on the computer?

Because it wanted to keep an eye on the mouse!

--- --- ---

Improved joke:
Why was the cat sitting on the computer?

Because it wanted to keep an eye on the mouse—after all, it heard they were great at clicking!

--- --- ---

Final joke:
Why was the cat sitting on the computer?

Because it wanted to keep an eye on the mouse—after all, it heard they were great at clicking! But little did it know, the mouse was actually an undercover tech whiz, programming a robot to serve catnip on demand!

Functional Version - Tasks

```
from langgraph.func import entrypoint, task

# Tasks
@task
def generate_joke(topic: str):
    """First LLM call to generate initial joke"""
    msg = llm.invoke(f"Write a short joke about {topic}")
    return msg.content

@task
def improve_joke(joke: str):
    """Second LLM call to improve the joke"""
    msg = llm.invoke(f"Make this joke funnier by adding wordplay: {joke}")
    return msg.content


@task
def polish_joke(joke: str):
    """Third LLM call for final polish"""
    msg = llm.invoke(f"Add a surprising twist to this joke: {joke}")
    return msg.content
```

Functional Version - Tasks

```
@task
def polish_joke(joke: str):
    """Third LLM call for final polish"""
    msg = llm.invoke(f"Add a surprising twist to this joke: {joke}")
    return msg.content
```

Unit of work

Executed asynchronously within an entrypoint

Return a future-like object that can be awaited or resolved synchronously

Logic & Invoking

```
def check_punchline(joke: str):
    """Gate function to check if the joke has a punchline"""
    # Simple check - does the joke contain "?" or "!"
    if "?" in joke or "!" in joke:
        return "Fail"

    return "Pass"

@entrypoint()
def parallel_workflow(topic: str):
    original_joke = generate_joke(topic).result()
    if check_punchline(original_joke) == "Pass":
        return original_joke

    improved_joke = improve_joke(original_joke).result()
    return polish_joke(improved_joke).result()

# Invoke
for step in parallel_workflow.stream("cats", stream_mode="updates"):
    print(step)
    print("\n")
```

Logic & Invoking

```
@entrypoint()
def parallel_workflow(topic: str):
    original_joke = generate_joke(topic).result()
    if check_punchline(original_joke) == "Pass":
        return original_joke

    improved_joke = improve_joke(original_joke).result()
    return polish_joke(improved_joke).result()
```

Entry Point

- Starting point of workflow
- Encapsulates logic
- Manages execution flow
  - Long-running tasks and interrupts

Output

{'generate_joke': 'Why was the cat sitting on the computer?\n\nBecause it wanted to keep an eye on the mouse!'}

{'improve_joke': 'Why was the cat sitting on the computer?\n\nBecause it wanted to keep an eye on the mouse and ensure it didn’t click away with all the cheese!'}

{'polish_joke': "Why was the cat sitting on the computer?\n\nBecause it wanted to keep an eye on the mouse, but then it discovered that the real cheese was all the secrets in the human's emails and decided to become a tech entrepreneur instead!"}

{'parallel_workflow': "Why was the cat sitting on the computer?\n\nBecause it wanted to keep an eye on the mouse, but then it discovered that the real cheese was all the secrets in the human's emails and decided to become a tech entrepreneur instead!"}

Joke App

Rag Joke App

Enter Joke Topic:

Chickens

Submit

Source

```
import streamlit as st
from langchain_openai import ChatOpenAI
from langgraph.func import entrypoint, task

openai_api_key = “XXX"
llm = ChatOpenAI(model="gpt-4.1-nano",openai_api_key=openai_api_key)

st.title("Rag Joke App")

def generate_response(input_text):
    for step in parallel_workflow.stream(input_text, stream_mode="updates"):
        st.info(step)

with st.form("my_form"):
    text = st.text_area("Enter Joke Topic:", "Chickens")
    submitted = st.form_submit_button("Submit")
    if submitted:
        generate_response(text)
```

```
# Tasks
@task
def generate_joke(topic: str):
    """First LLM call to generate initial joke"""
    msg = llm.invoke(f"Write a short joke about {topic}")
    return msg.content


def check_punchline(joke: str):
    """Gate function to check if the joke has a punchline"""
    # Simple check - does the joke contain "?" or "!"
    if "?" in joke or "!" in joke:
        return "Fail"
    return "Pass"


@task
def improve_joke(joke: str):
    """Second LLM call to improve the joke"""
    msg = llm.invoke(f"Make this joke funnier by adding wordplay: {joke}")
    return msg.content


@task
def polish_joke(joke: str):
    """Third LLM call for final polish"""
    msg = llm.invoke(f"Add a surprising twist to this joke: {joke}")
    return msg.content


@entrypoint()
def parallel_workflow(topic: str):
    original_joke = generate_joke(topic).result()
    if check_punchline(original_joke) == "Pass":
        return original_joke

    improved_joke = improve_joke(original_joke).result()
    return polish_joke(improved_joke).result()
```

Entry Point - checkpointer

Enables Persistence

```
from langgraph.func import entrypoint
from langgraph.checkpoint.memory import MemorySaver
from typing import Any

@entrypoint(checkpointer=MemorySaver())
def my_workflow(number: int, *, previous: Any = None) -> int:
    previous = previous or 0
    return number + previous

config = {
    "configurable": {
        "thread_id": "some_thread_id"
    }
}

my_workflow.invoke(1, config)  # 1 (previous was None)
my_workflow.invoke(2, config)  # 3 (previous was 1 from the previous invocation)
```

Injectable Parameters

Table 1

Parameter	Description
previous	Access the the state associated with the previous checkpoint for the given thread.
store	Useful for long-term memory.
writer	For streaming custom data, to write custom data to the custom stream.
config	For accessing run time configuration.

Injectable Parameters

Table 1

Parameter	Description
previous	Access the the state associated with the previous checkpoint for the given thread.
store	Useful for long-term memory.
writer	For streaming custom data, to write custom data to the custom stream.
config	For accessing run time configuration.

```
from langchain_core.runnables import RunnableConfig
from langgraph.func import entrypoint
from langgraph.store.base import BaseStore
from langgraph.store.memory import InMemoryStore

in_memory_store = InMemoryStore(...)  # An instance of InMemoryStore for long-term memory

@entrypoint(
    checkpointer=checkpointer,  # Specify the checkpointer
    store=in_memory_store  # Specify the store
)
def my_workflow(
    some_input: dict,  # The input (e.g., passed via `invoke`)
    *,
    previous: Any = None, # For short-term memory
    store: BaseStore,  # For long-term memory
    writer: StreamWriter,  # For streaming custom data
    config: RunnableConfig  # For accessing the configuration passed to the entrypoint
) -> ...:
```

```
@entrypoint(checkpointer=checkpointer)
def my_workflow(number: int, *, previous: Any = None) -> int:
    previous = previous or 0
    return number + previous

config = {
    "configurable": {
        "thread_id": "some_thread_id"
    }
}

my_workflow.invoke(1, config)  # 1 (previous was None)
my_workflow.invoke(2, config)  # 3 (previous was 1 from the previous invocation)
```