CS 696 Applied Large Language Models
Spring Semester, 2025
Doc 20 News, LangChain, Version 2
Apr 10, 2025

OpenAI Reasoning Models

Reasoning models (o1 and o3-mini) vs GPT models (like GPT-4o)

They behave differently

o-series models

Think longer and harder about complex tasks

Execute tasks with high accuracy and precision

Use o-series models to plan out the strategy to solve a problem, and use GPT models to execute specific tasks

How to choose

Speed and cost → GPT models are faster and tend to cost less

Executing well-defined tasks → GPT models handle explicitly defined tasks well

Accuracy and reliability → o-series models are reliable decision-makers

Complex problem-solving → o-series models work through ambiguity and complexity

https://platform.openai.com/docs/guides/reasoning-best-practices

When to use reasoning models

Navigating ambiguous tasks

- Good at
  - Taking limited information
  - Disparate pieces of information
  - With a simple prompt,
    - Understanding the user’s intent
    - Handling any gaps in the instructions

Finding a needle in a haystack

- Good at
  - Understanding and pulling out only the most relevant information
  - From large amounts of unstructured information

When to use reasoning models

Finding relationships and nuance across a large dataset

Multi-step agentic planning

- Reasoning model creates plan
- Selects which model to do each step

Visual reasoning

Reviewing, debugging, and improving code quality

Evaluation and benchmarking for other model responses

How to prompt reasoning models effectively

Developer messages replace system messages

Keep prompts simple and direct

Avoid chain-of-thought prompts

Use delimiters for clarity

  Clearly indicate distinct parts of the input

Try zero shot first, then few shot if needed

Provide specific guidelines

Be very specific about your end goal

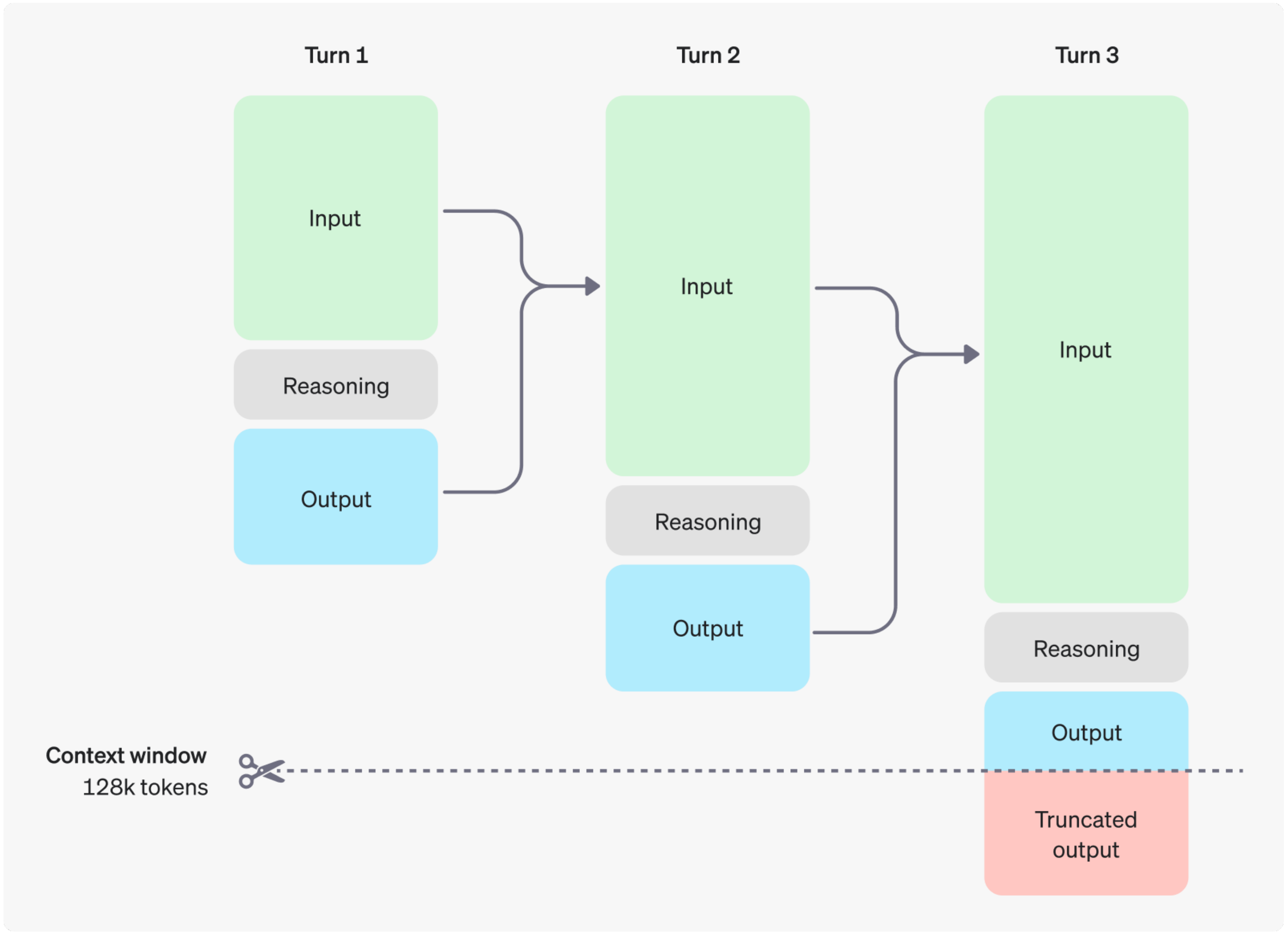
Markdown formatting

  Will not use markdown in API unless requested

  Formatting re-enabled on first line of developer message

Reasoning Tokens

Reasoning models add reasoning tokens



https://platform.openai.com/docs/guides/reasoning/advice-on-prompting?api-mode=chat

Tracing the thoughts of a large language model

https://www.anthropic.com/research/tracing-thoughts-language-model, Mar 27, 2025

Garçon

  Tool to observe inner workings of model

Claude sometimes thinks in a conceptual space that is shared between languages

Claude will plan what it will say many words ahead

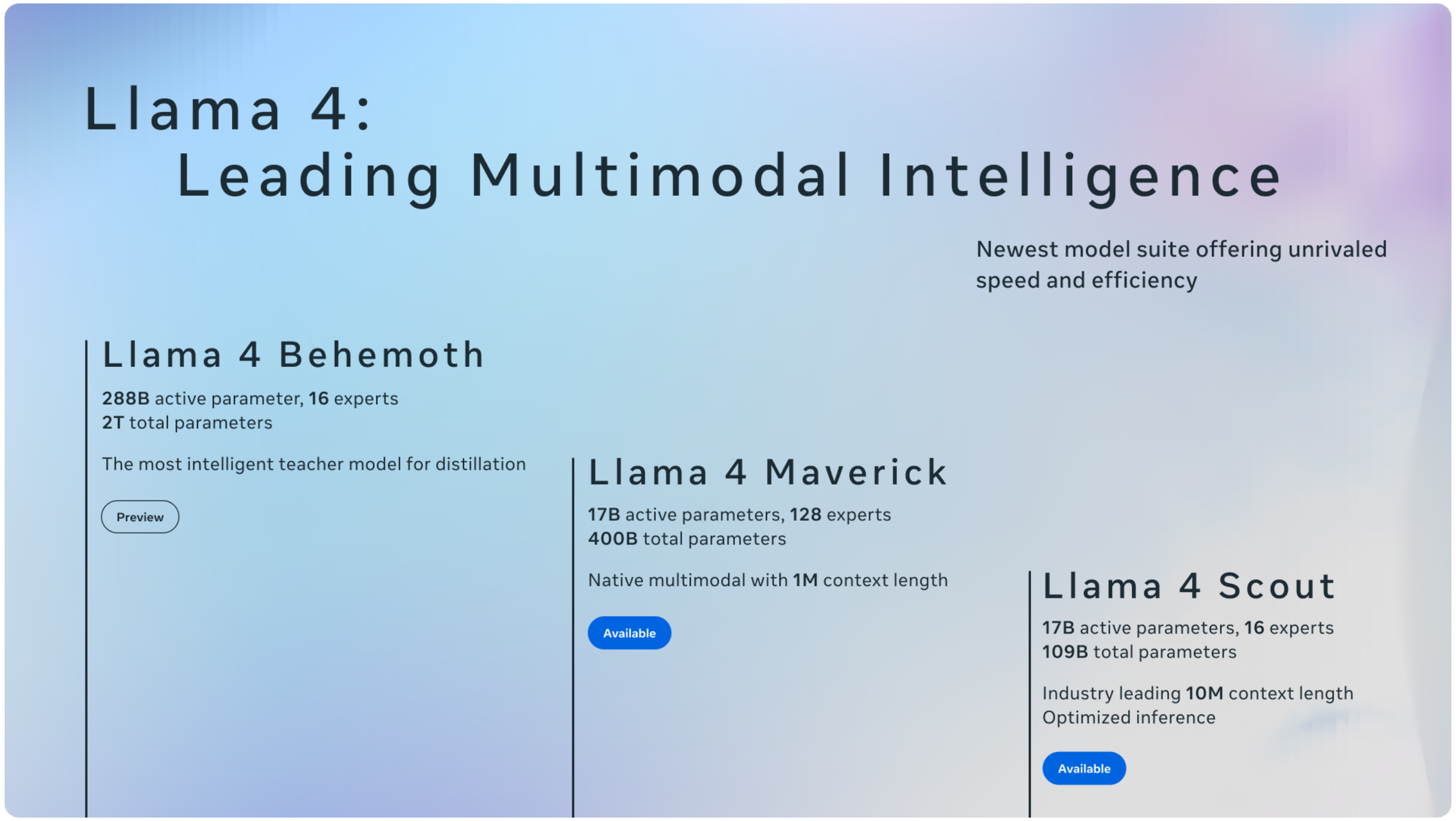
Claude, on occasion, will give a plausible-sounding argument designed to agree with the user

TransformerLens

https://github.com/TransformerLensOrg/TransformerLens

Library for doing mechanistic interpretability of GPT-2 Style language models

Llama 4



Llama 4:
Leading Multimodal Intelligence

Newest model suite offering unrivaled speed and efficiency

Llama 4 Behemoth

288B active parameter, **16** experts
2T total parameters

The most intelligent teacher model for distillation

Preview

Llama 4 Maverick

17B active parameters, **128** experts
400B total parameters

Native multimodal with **1M** context length

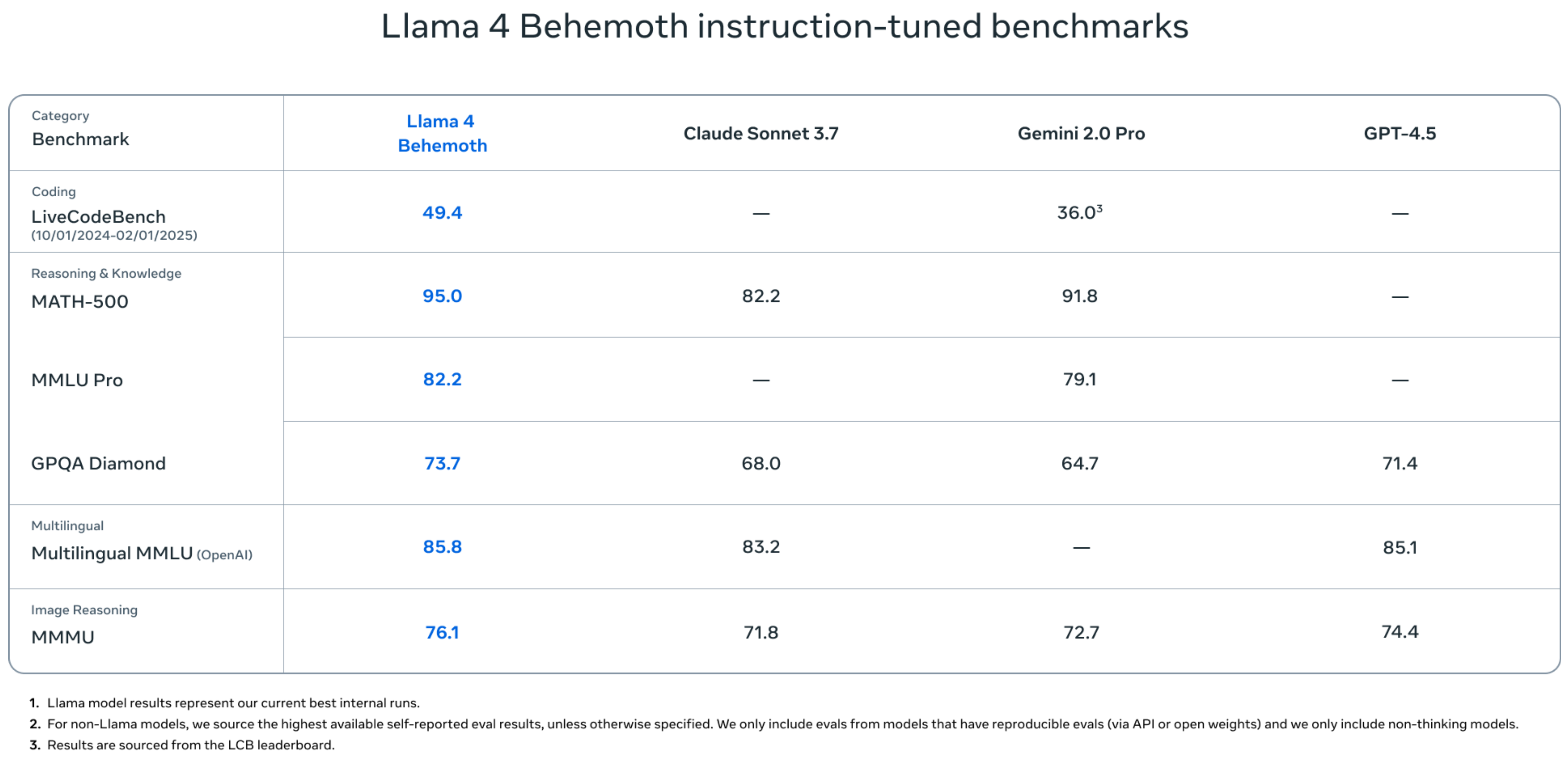
Available

Llama 4 Scout

17B active parameters, **16** experts
109B total parameters

Industry leading **10M** context length
Optimized inference

https://ai.meta.com/blog/llama-4-multimodal-intelligence/

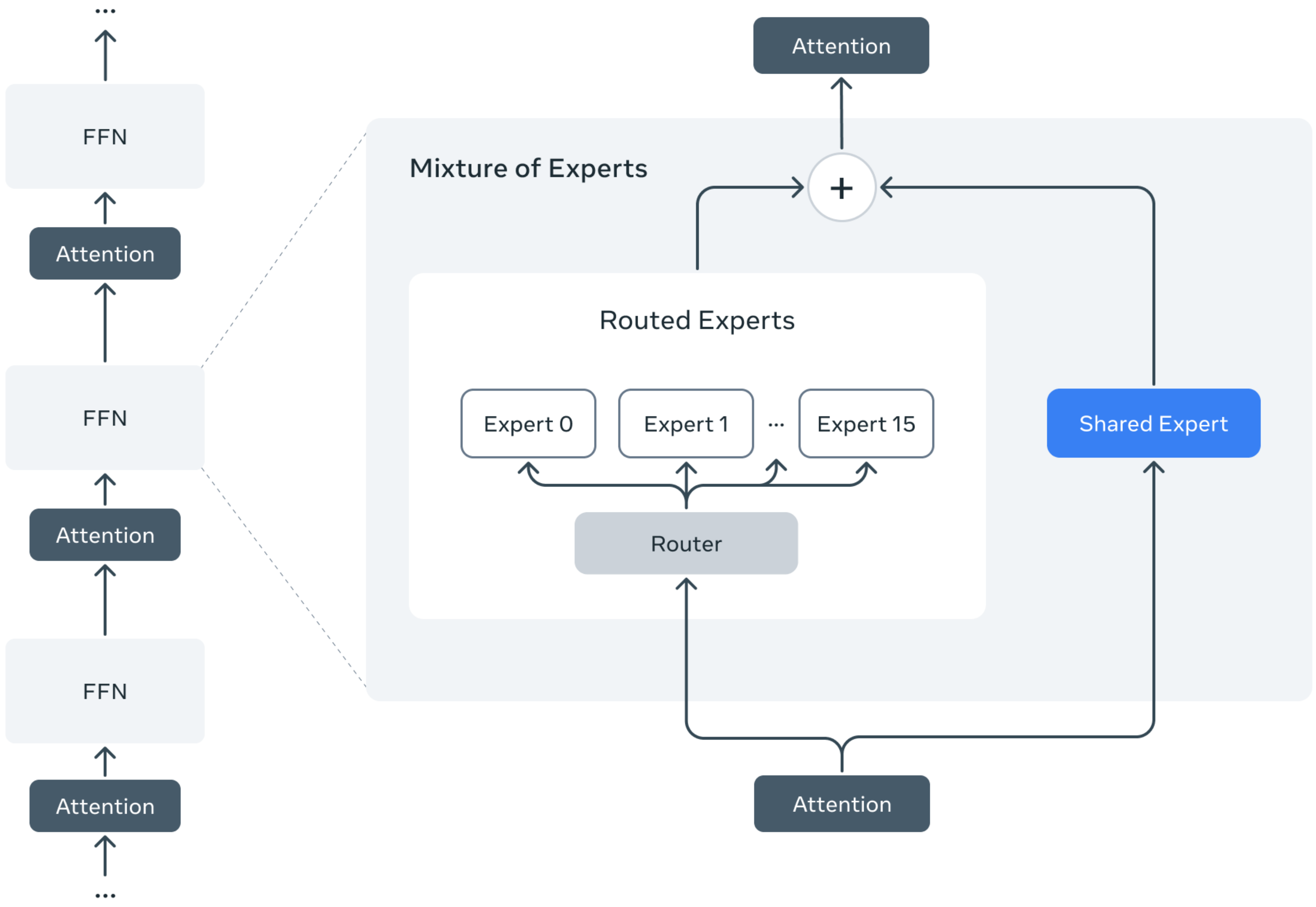


Llama 4 Behemoth instruction-tuned benchmarks

Category Benchmark	Llama 4 Behemoth	Claude Sonnet 3.7	Gemini 2.0 Pro	GPT-4.5
Coding LiveCodeBench (10/01/2024-02/01/2025)	**49.4**	—	36.0[3]	—
Reasoning & Knowledge MATH-500	**95.0**	82.2	91.8	—
MMLU Pro	**82.2**	—	79.1	—
GPQA Diamond	**73.7**	68.0	64.7	71.4
Multilingual Multilingual MMLU (OpenAI)	**85.8**	83.2	—	85.1
Image Reasoning MMMU	**76.1**	71.8	72.7	74.4

1. Llama model results represent our current best internal runs.
2. For non-Llama models, we source the highest available self-reported eval results, unless otherwise specified. We only include evals from models that have reproducible evals (via API or open weights) and we only include non-thinking models.
3. Results are sourced from the LCB leaderboard.

https://ai.meta.com/blog/llama-4-multimodal-intelligence/



https://huggingface.co/blog/moe

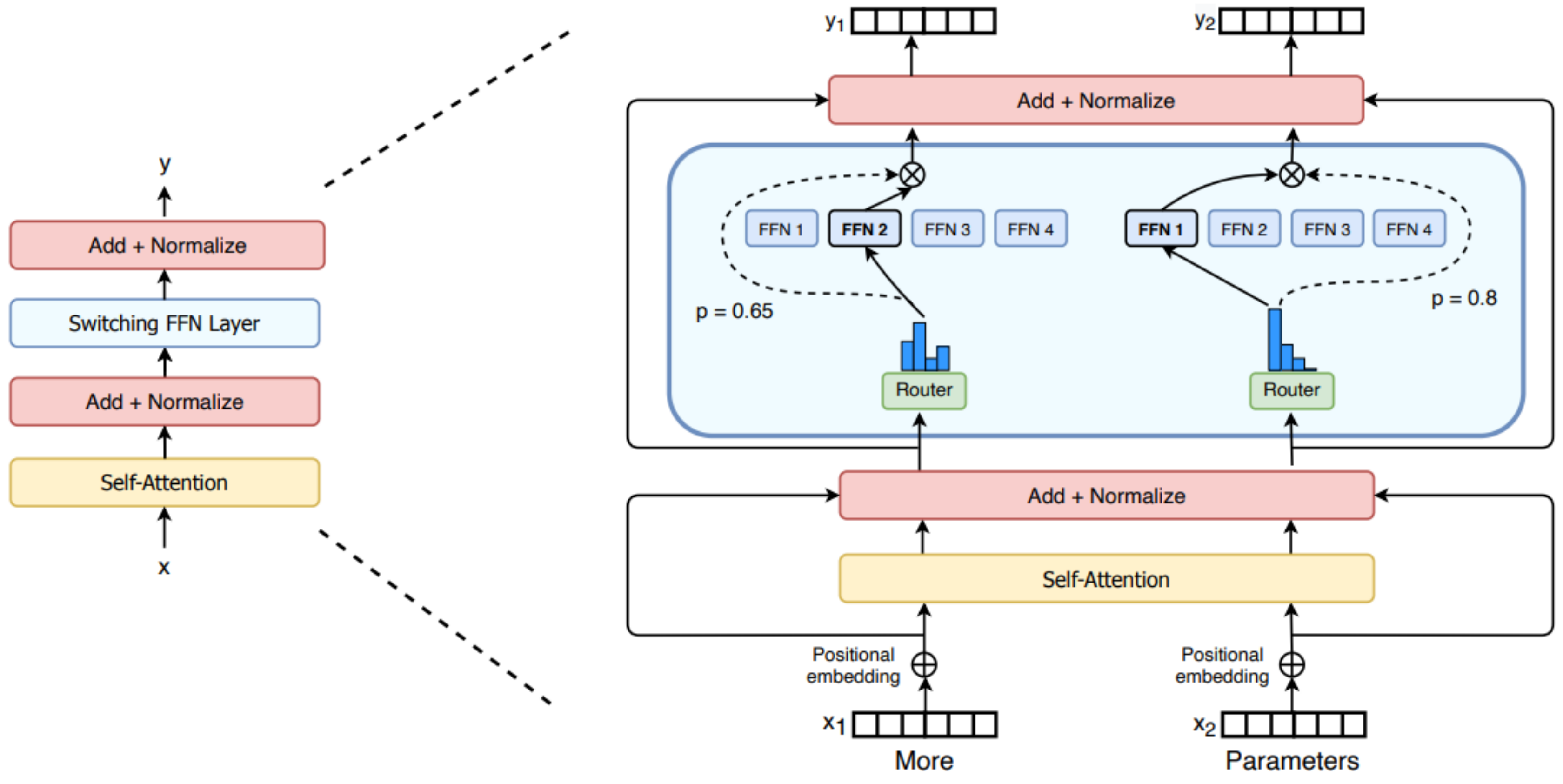


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

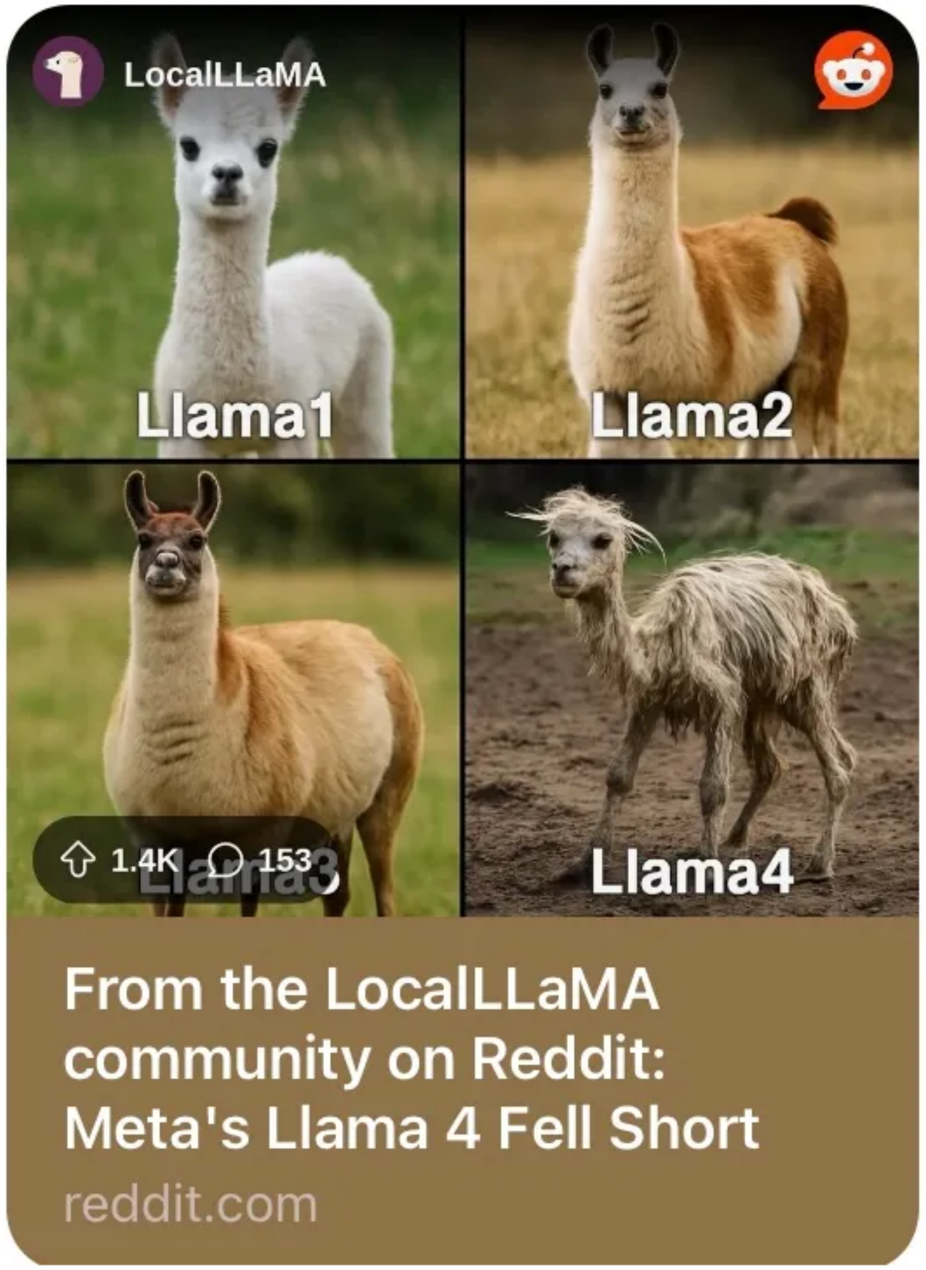
Deep Learning, Deep Scandal

Apr 7, 2025

“Deep learning is indeed finally hitting a wall, in the sense of reaching a point of diminishing results.”

Meta did an experiment, and the experiment didn’t work; that’s science. The idea that you could predict a model’s performance entirely according to its size and the size of its data just turns out to be wrong

According to a rumor that sounds pretty plausible, the powers-that-be at Meta weren’t happy with the results, and wanted something better badly enough that they may have tried to cheat, per a thread on reddit (original in Chinese):



From the LocalLLaMA community on Reddit: Meta's Llama 4 Fell Short
reddit.com

https://garymarcus.substack.com/p/deep-learning-deep-scandal

Chatbots Are Cheating on Their Benchmark Tests

March 5, 2025

“AI programs train on questions they’re later tested on. So how do we know if they’re getting smarter?”

“Yet there is growing evidence that progress is slowing down and that the LLM-powered chatbot may already be near its peak. This is troubling, given that the promise of advancement has become a political issue; massive amounts of land, power, and money have been earmarked to drive the technology forward.”

https://www.theatlantic.com/technology/archive/2025/03/chatbots-benchmark-tests/681929/

Recent AI model progress feels mostly like bullshit

24th Mar 2025

Started a cybersecurity company using AI

Since 3.5-sonnet monitor models
- 3.6 minor improvement
- 3.7 smaller improvement
- New models have no noticeable improvement

But I would nevertheless like to submit, based off of internal benchmarks, and my own and colleagues' perceptions using these models, that whatever gains these companies are reporting to the public, they are not reflective of economic usefulness or generality. They are not reflective of my Lived Experience or the Lived Experience of my customers

https://www.lesswrong.com/posts/4mvphwx5pdsZLMmpY/recent-ai-model-progress-feels-mostly-like-bullshit

Benchmarks

Code & data are publicly available

How do we know if models are accidentally trained on some of the data

How to know if company purposely trained on benchmark data

Do benchmarks measure anything meaningful

Still Full Speed Ahead

Shopify CEO

"demonstrate why they cannot get what they want done using AI" before requesting additional headcount or resources.

Use of AI will now be a component of their performance reviews

Cisco EVP

This note from Tobi to his employees at Shopify isn't much different from what we have been discussing at Cisco for the past several months

There will only be two kinds of companies that will exist in the future. Those that will be AI-forward companies and others who will discount AI and struggle for relevance

Andriy Burkov, AI PhD, Author

An adequate CEO would say, "You should use AI whenever you feel it makes you more productive and our company richer. Otherwise, use the most appropriate tool at your disposal."

https://www.linkedin.com/news/story/shopify-ceo-issues-ai-ultimatum-6713809/

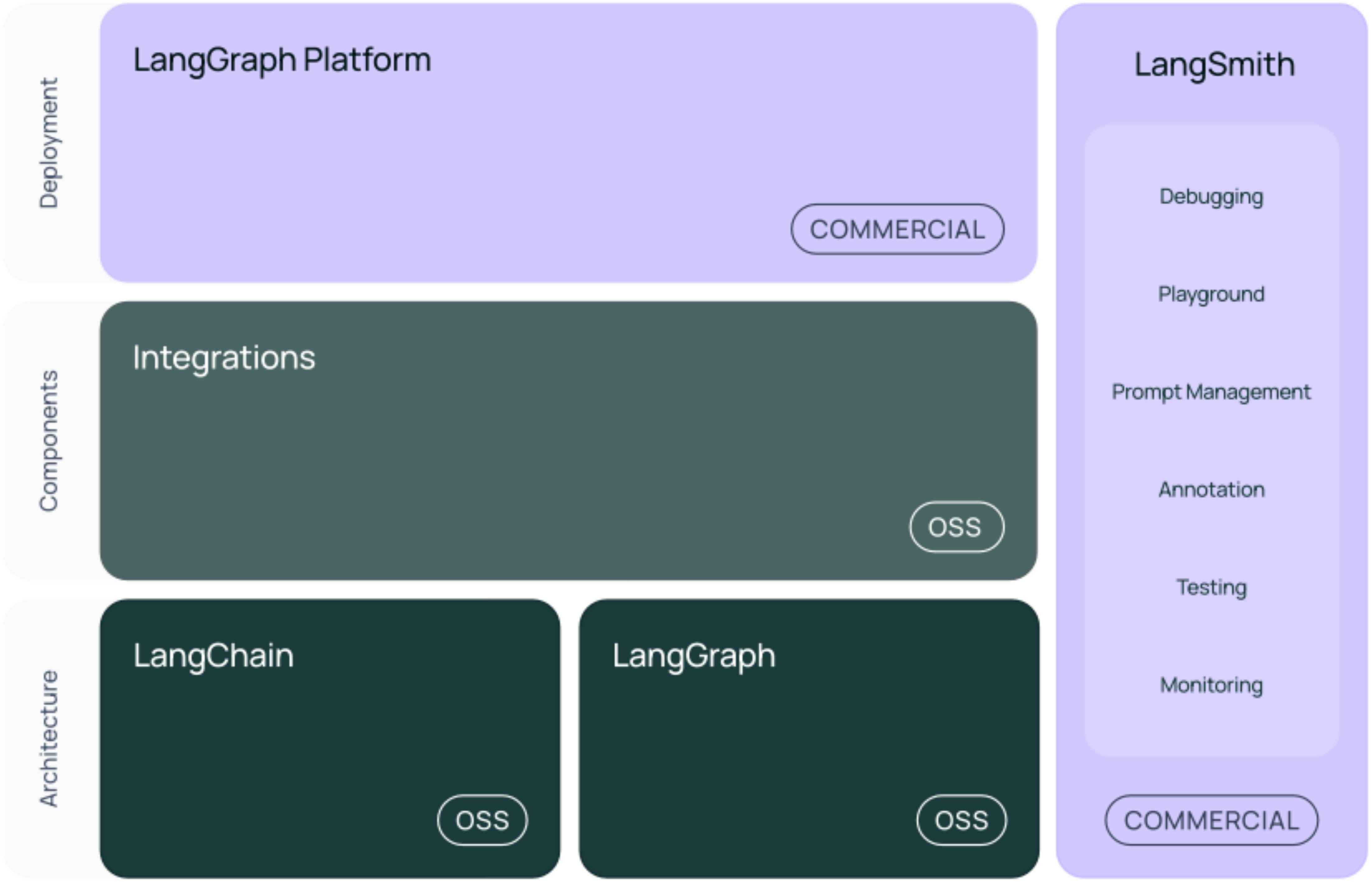
LangChain, LangSmith, LangGraph

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


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

Hello World

```
LANGSMITH_TRACING=True
LANGSMITH_ENDPOINT="https://api.smith.langchain.com"
LANGSMITH_API_KEY="XXX"
LANGSMITH_PROJECT="Hello World"
OPENAI_API_KEY=“YYY"
```

Hello World

```
import getpass
import os

try:
    # load environment variables from .env file (requires `python-dotenv`)
    from dotenv import load_dotenv

    load_dotenv()
except ImportError:
    pass

os.environ["LANGSMITH_TRACING"] = "true"
if "LANGSMITH_API_KEY" not in os.environ:
    os.environ["LANGSMITH_API_KEY"] = LANGSMITH_API_KEY
if "LANGSMITH_PROJECT" not in os.environ:
    os.environ["LANGSMITH_PROJECT"] = LANGSMITH_PROJECT
    if not os.environ.get("LANGSMITH_PROJECT"):
        os.environ["LANGSMITH_PROJECT"] = "default"
if "OPENAI_API_KEY" not in os.environ:
    os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY
```

Hello World

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI()
llm.invoke("Hello, world!")
```

```
AIMessage(content='Hello! How can I assist you today?',
    additional_kwargs={'refusal': None},
    response_metadata={'token_usage': {'completion_tokens': 10, 'prompt_tokens': 11, 'total_tokens': 21,
      'completion_tokens_details': {'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0,
      'rejected_prediction_tokens': 0},
    ‘prompt_tokens_details': {'audio_tokens': 0, 'cached_tokens': 0}},
    'model_name': 'gpt-3.5-turbo-0125',
    'system_fingerprint': None,
    'id': 'chatcmpl-BJUBxyFpKqjiBDEtmfKcjpQb0gAgo',
    'finish_reason': 'stop', 'logprobs': None},
    id='run-722f118d-c2f6-44a3-bff6-b05d353256f8-0',
    usage_metadata={'input_tokens': 11, 'output_tokens': 10, 'total_tokens': 21,
    'input_token_details': {'audio': 0, 'cache_read': 0},
    'output_token_details': {'audio': 0, 'reasoning': 0}})
```

Why Environment Variables

Messages

```
from langchain.chat_models import init_chat_model
from langchain_core.messages import HumanMessage, SystemMessage

model = init_chat_model("gpt-4o-mini", model_provider="openai")

messages = [
    SystemMessage("Translate the following from English into Italian"),
    HumanMessage("hi!"),
]

model.invoke(messages)
```

Messages

- Roles
  - System
  - User
  - Assistant
    - Model Response
  - Tool
    - Pass the results of a tool invocation back to the model after external data or processing has been retrieved

- Content
  - SystemMessage
  - HumanMessage
  - AIMessage

ChatPromptTemplate

```
from langchain_core.prompts import ChatPromptTemplate

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)
print(response.content)
```

SystemMessage

No standard among models

Different models use different keywords

LangChain maps SystemMessge to the model’s required keyword in most cases

Done if using a LangChain chat model

LangChain Supported Models

’openai’ -> langchain-openai

’anthropic’ -> langchain-anthropic

’azure_openai’ -> langchain-openai

’azure_ai’ -> langchain-azure-ai

’google_vertexai’ -> langchain-google-vertexai

’google_genai’ -> langchain-google-genai

’bedrock’ -> langchain-aws

’bedrock_converse’ -> langchain-aws

’cohere’ -> langchain-cohere

’fireworks’ -> langchain-fireworks

’together’ -> langchain-together

’mistralai’ -> langchain-mistralai

’huggingface’ -> langchain-huggingface

’groq’ -> langchain-groq

’ollama’ -> langchain-ollama

’google_anthropic_vertex’ -> langchain-google-vertexai

’deepseek’ -> langchain-deepseek

’ibm’ -> langchain-ibm

’nvidia’ -> langchain-nvidia-ai-endpoints

’xai’ -> langchain-xai

’perplexity’ -> langchain-perplexity

Remember the Prompt

~~~
prompt_text = """You are an expert AI assistant that explains your reasoning step by step. For each step, provide a title that describes what you're doing in that step, along with the content. Decide if you need another step or if you're ready to give the final answer. Respond in JSON format with 'title', 'content', and 'next_action' (either 'continue' or 'final_answer') keys. USE AS MANY REASONING STEPS AS POSSIBLE. AT LEAST 3. BE AWARE OF YOUR LIMITATIONS AS AN LLM AND WHAT YOU CAN AND CANNOT DO. IN YOUR REASONING, INCLUDE EXPLORATION OF ALTERNATIVE ANSWERS. CONSIDER YOU MAY BE WRONG, AND IF YOU ARE WRONG IN YOUR REASONING, WHERE IT WOULD BE. FULLY TEST ALL OTHER POSSIBILITIES. YOU CAN BE WRONG. WHEN YOU SAY YOU ARE RE-EXAMINING, ACTUALLY RE-EXAMINE, AND USE ANOTHER APPROACH TO DO SO. DO NOT JUST SAY YOU ARE RE-EXAMINING. USE AT LEAST 3 METHODS TO DERIVE THE ANSWER. USE BEST PRACTICES.

Example of a valid JSON response:
```json
{
    "title": "Identifying Key Information",
    "content": "To begin solving this problem, we need to carefully examine the given information and identify the crucial elements that will guide our solution process. This involves...",
    "next_action": "continue"
}```
"""
~~~

Fail

```
model = init_chat_model("gpt-4o-mini", model_provider="openai")

prompt_template = ChatPromptTemplate.from_messages(
    [("system", prompt_text), ("user", "{foo}")]
)
prompt = prompt_template.invoke({ "foo": "How many Rs are in strawberry?"})
response = model.invoke(prompt)
print(response.content)
```

```
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 '
```

Corrected Prompt

~~~
prompt_text = """You are an expert AI assistant that explains your reasoning step by step. For each step, provide a title that describes what you're doing in that step, along with the content. Decide if you need another step or if you're ready to give the final answer. Respond in JSON format with 'title', 'content', and 'next_action' (either 'continue' or 'final_answer') keys. USE AS MANY REASONING STEPS AS POSSIBLE. AT LEAST 3. BE AWARE OF YOUR LIMITATIONS AS AN LLM AND WHAT YOU CAN AND CANNOT DO. IN YOUR REASONING, INCLUDE EXPLORATION OF ALTERNATIVE ANSWERS. CONSIDER YOU MAY BE WRONG, AND IF YOU ARE WRONG IN YOUR REASONING, WHERE IT WOULD BE. FULLY TEST ALL OTHER POSSIBILITIES. YOU CAN BE WRONG. WHEN YOU SAY YOU ARE RE-EXAMINING, ACTUALLY RE-EXAMINE, AND USE ANOTHER APPROACH TO DO SO. DO NOT JUST SAY YOU ARE RE-EXAMINING. USE AT LEAST 3 METHODS TO DERIVE THE ANSWER. USE BEST PRACTICES.

Example of a valid JSON response:
```json
{{
    "title": "Identifying Key Information",
    "content": "To begin solving this problem, we need to carefully examine the given information and identify the crucial elements that will guide our solution process. This involves...",
    "next_action": "continue"
}}```
"""
~~~

Now Works

```
model = init_chat_model("gpt-4o-mini", model_provider="openai")

prompt_template = ChatPromptTemplate.from_messages(
    [("system", prompt_text), ("user", "{foo}")]
)
prompt = prompt_template.invoke({ "foo": "How many Rs are in strawberry?"})
response = model.invoke(prompt)
print(response.content)
```

Output

~~~
```json
{  "title": "Understanding the Question",
    "content": "The question asks how many 'Rs' are in the word 'strawberry'. To answer this, I will count the occurrences of the letter 'R' in the spelling of the word.",
    "next_action": "continue"
}
```
```json
{ "title": "Analyzing the Word",
    "content": "The word 'strawberry' is spelled as follows: s-t-r-a-w-b-e-r-r-y. I will check each letter in the spelling to find the letter 'R'.",
    "next_action": "continue"
}
```
```json
{  "title": "Counting the Occurrences",
    "content": "In 'strawberry', the letter 'R' appears twice: once as the fifth letter and once as the sixth letter. Therefore, the total count of 'R's in the word is 2.",
    "next_action": "final_answer"
}
```
```json
{  "title": "Final Answer",
    "content": "There are 2 Rs in the word 'strawberry'.",
    "next_action": "final_answer"
}
```
~~~

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!"})
```

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)
```

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, 0.02107970230281353,
-0.009436433320351839, 0.0713602676987648, -0.0028291644994169474,
0.02665858529508114]
```

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: 0.48980270038282736

Score: 0.43804432500253065
```

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)
```

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)

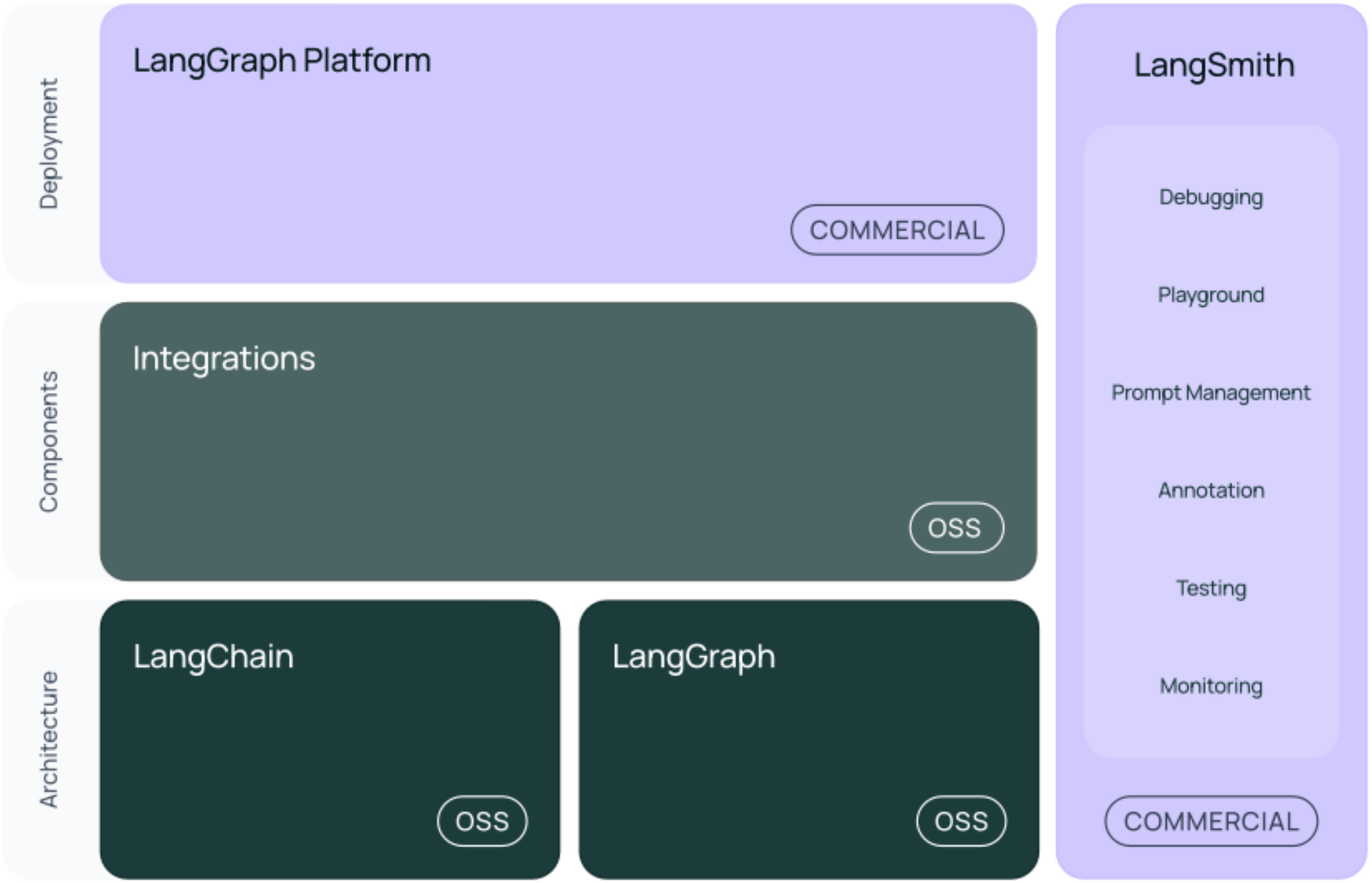
Embedding Search

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

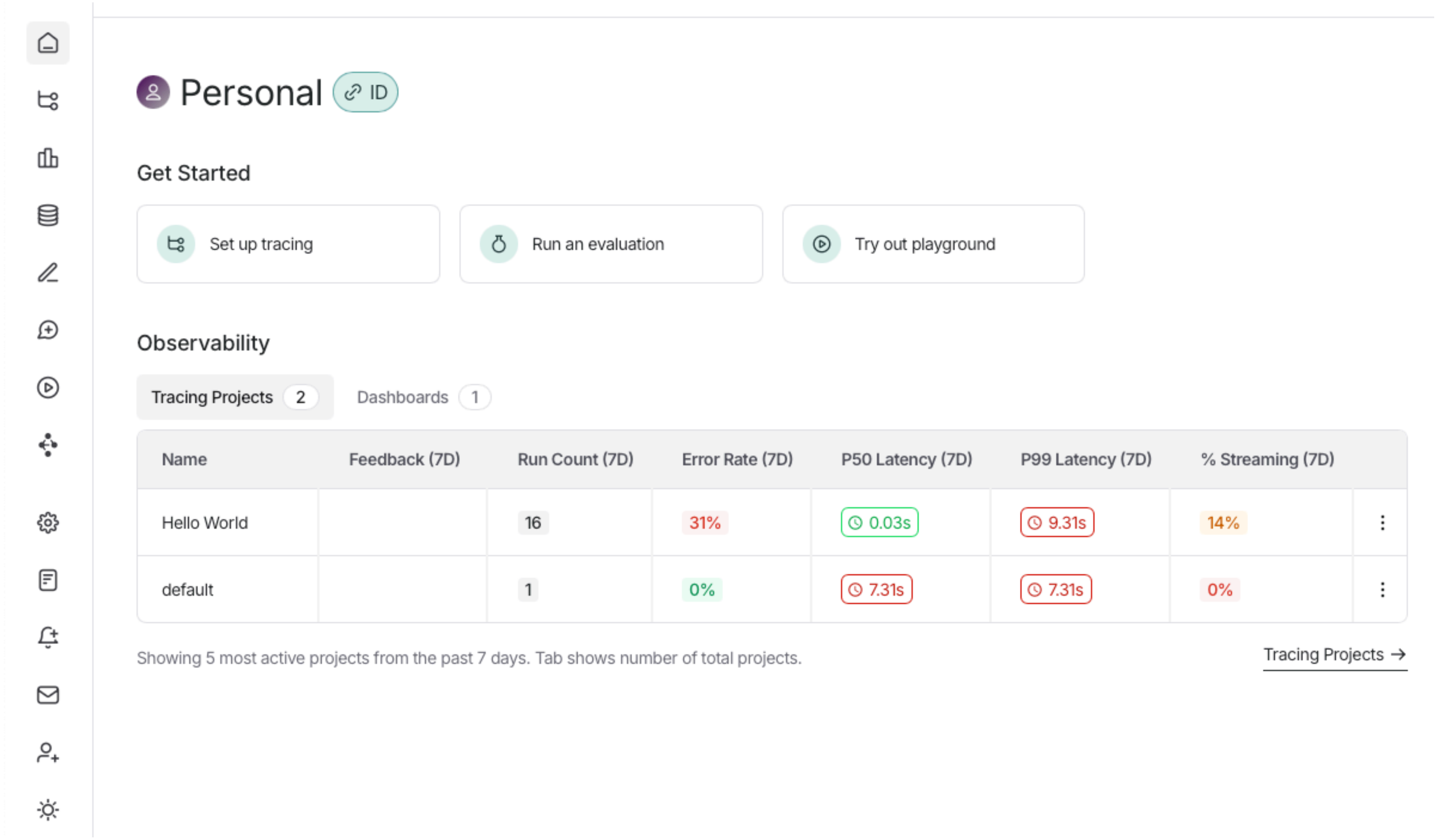
Saving and Loading

```
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



Personal ID

Get Started

Set up tracing

Run an evaluation

Try out playground

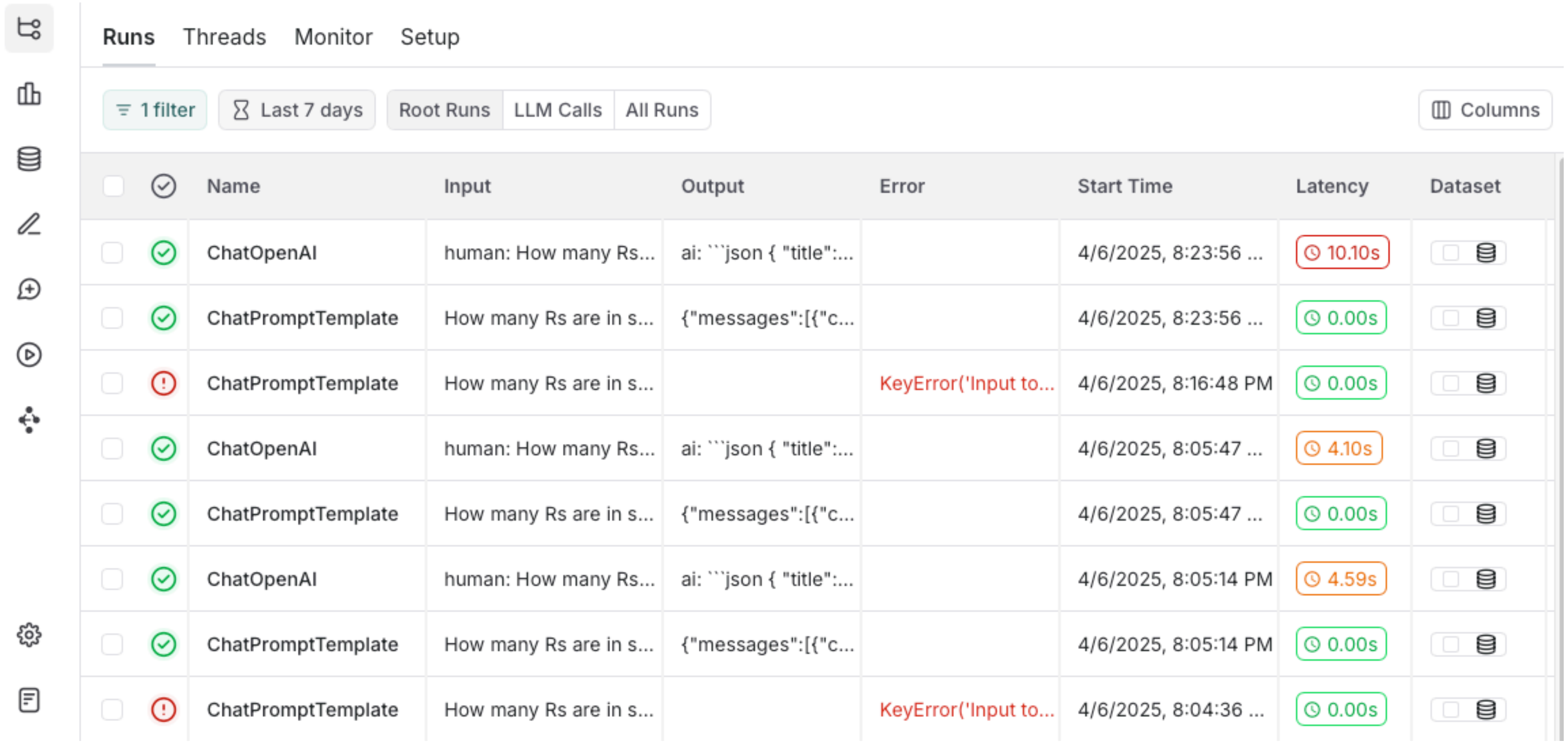
Observability

Tracing Projects 2 | Dashboards 1

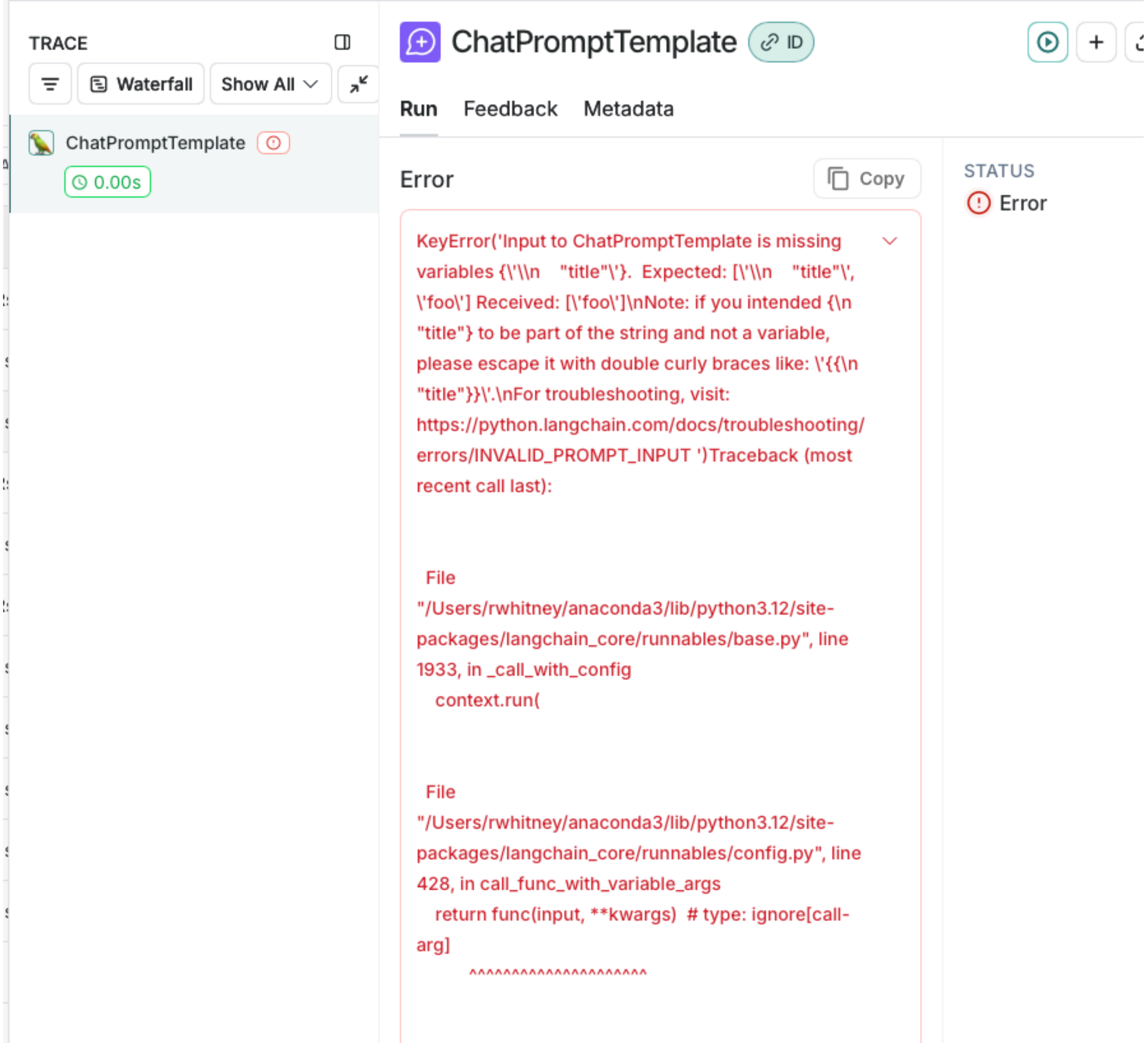
Name	Feedback (7D)	Run Count (7D)	Error Rate (7D)	P50 Latency (7D)	P99 Latency (7D)	% Streaming (7D)
Hello World		16	31%	0.03s	9.31s	14%
default		1	0%	7.31s	7.31s	0%

Showing 5 most active projects from the past 7 days. Tab shows number of total projects.

Tracing Projects →



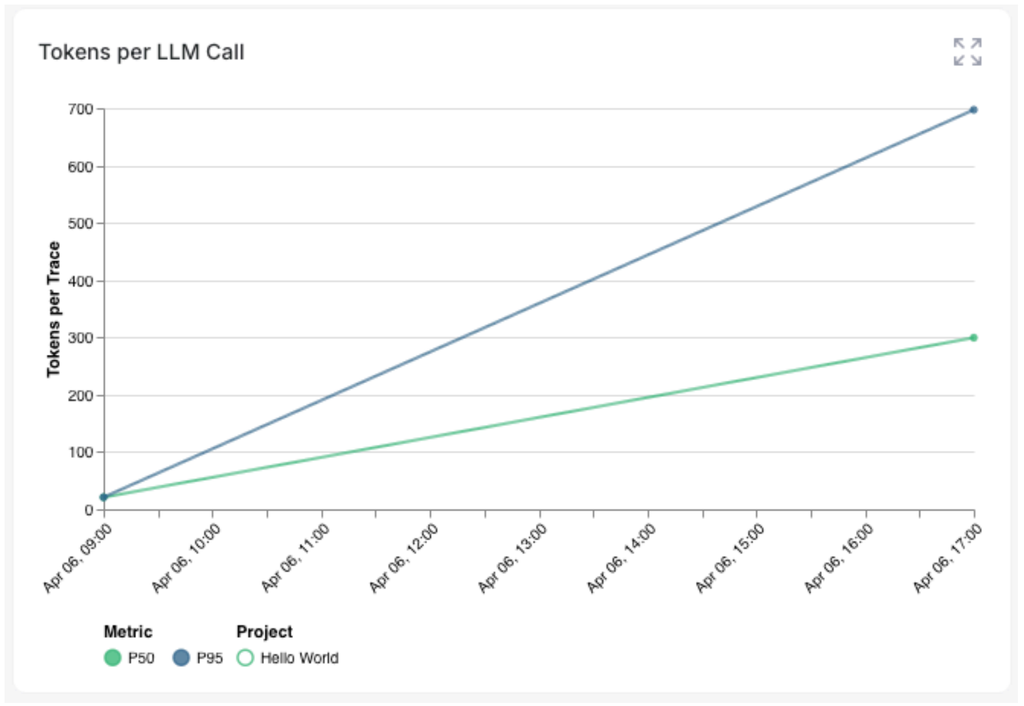
		Name	Input	Output	Error	Start Time	Latency	Dataset
	✓	ChatOpenAI	human: How many Rs...	ai: ```json { "title":...		4/6/2025, 8:23:56 ...	10.10s	
	✓	ChatPromptTemplate	How many Rs are in s...	{"messages":[{"c...		4/6/2025, 8:23:56 ...	0.00s	
	!	ChatPromptTemplate	How many Rs are in s...		KeyError('Input 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	

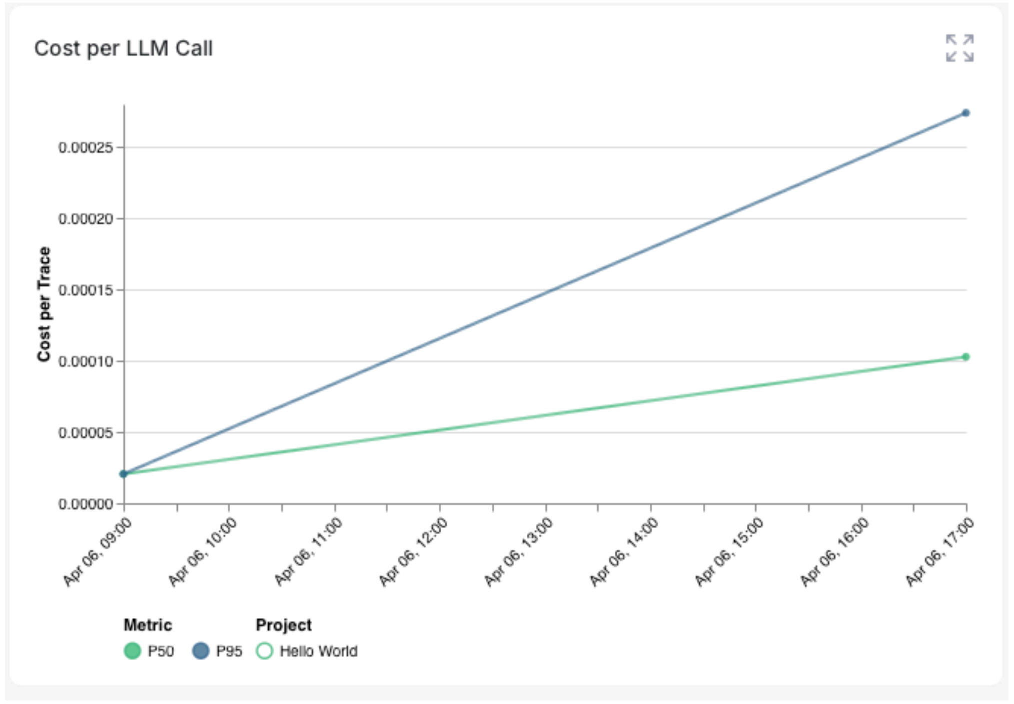


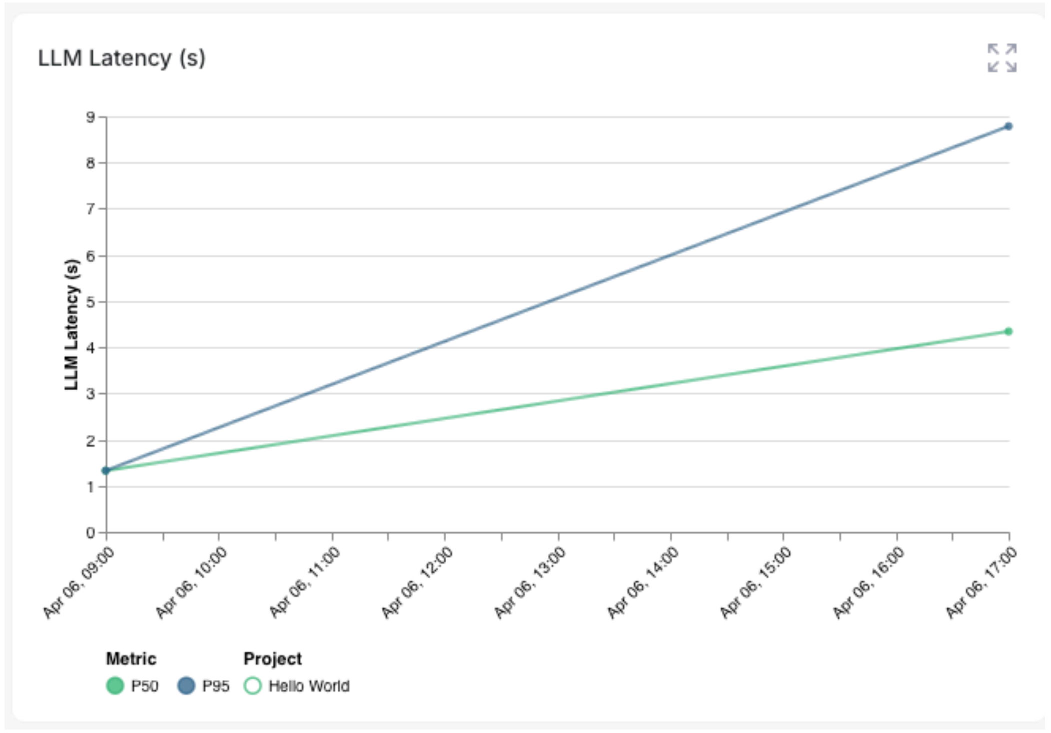
Error

```
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]
           ^^^^^^^^^^^^^^^^^^^^^
```

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
● P50
● P95
Project
○ 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
● P50
● P95
Project
○ 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
● P50
● P95
Project
○ 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

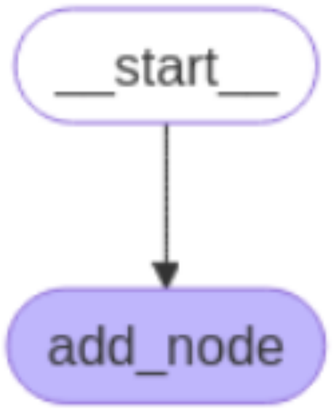
First Example

```
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

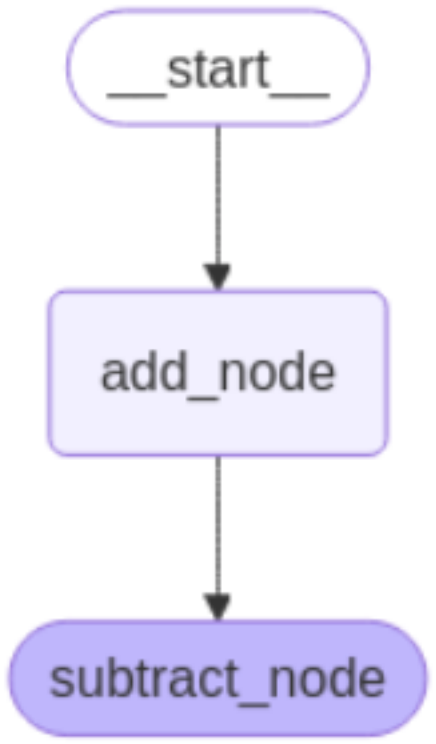
```
from langgraph.graph import START, StateGraph

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
```

```
{'x': -18}
```

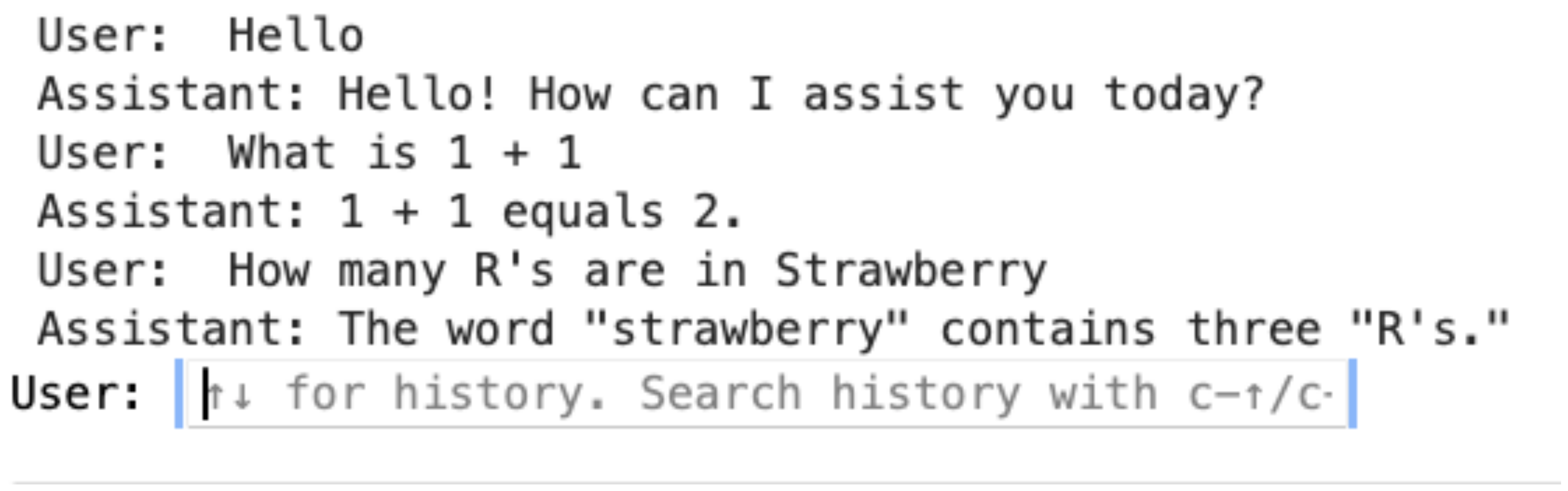


First Example

```
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-
```