Monitoring and Evaluating LLM Apps

Using Langfuse

Ruan Pretorius | October 2024 | melio.ai

About me

Hi there, I'm Ruan Pretorius 👋

- ☕ I turn coffee into AI
- 🖥️ I am a data scientist at *melio.ai*
  - We help you build and deploy your data intensive apps to unlock value from your data, follow us on LinkedIn
- 🔗 You can find me on GitHub *@ruankie*
- ✉️ Or contact me via email: *ruan@melio.ai*

melio.ai

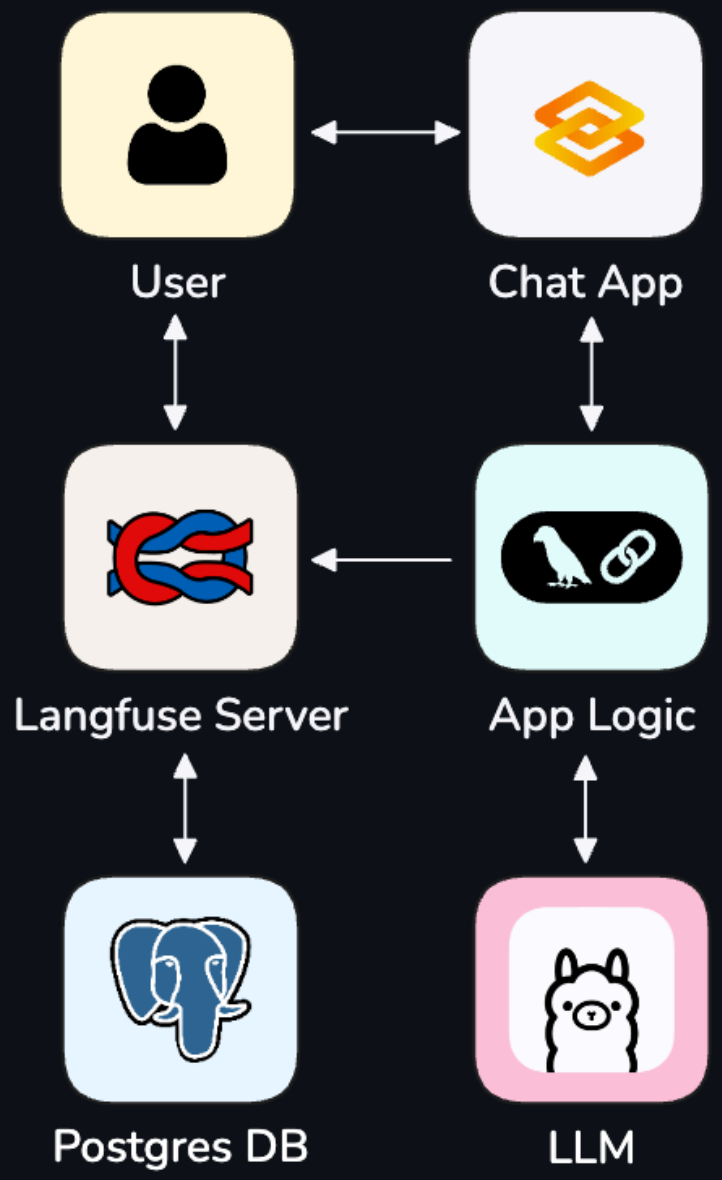
📌 Outline

- **Introduction**: Why monitor/evaluate LLM apps?
- **Setup**: Local LLMs for prototyping
- **Monitoring**: with Langfuse
- **Evaluation**: LLM-assisted with Ollama and Langfuse
- **Takeaways and Conclusion**

❓ Why Monitor & Evaluate LLM Apps?

- **Ensure Quality & Performance**
  - Track hallucination, retrieval accuracy, latency, etc.
  - To maintain a high-quality user experience
- **Detect Errors**
  - Harmful outputs
- **Identify Areas of Improvement**
  - Reduce costs
  - Reduce latency
  - Improve answers
  - If failure occurs, see when and where

What We'll Be Building



⚡ Setting Up

> Local LLMs for zero-cost learning and prototyping

🦙 Ollama

- For locally running LLMs
- Available for macOS, Linux, and Windows (preview)
- Familiar Docker feel with `:version` tags and commands like `pull` and `run`

🦙 Ollama Setup

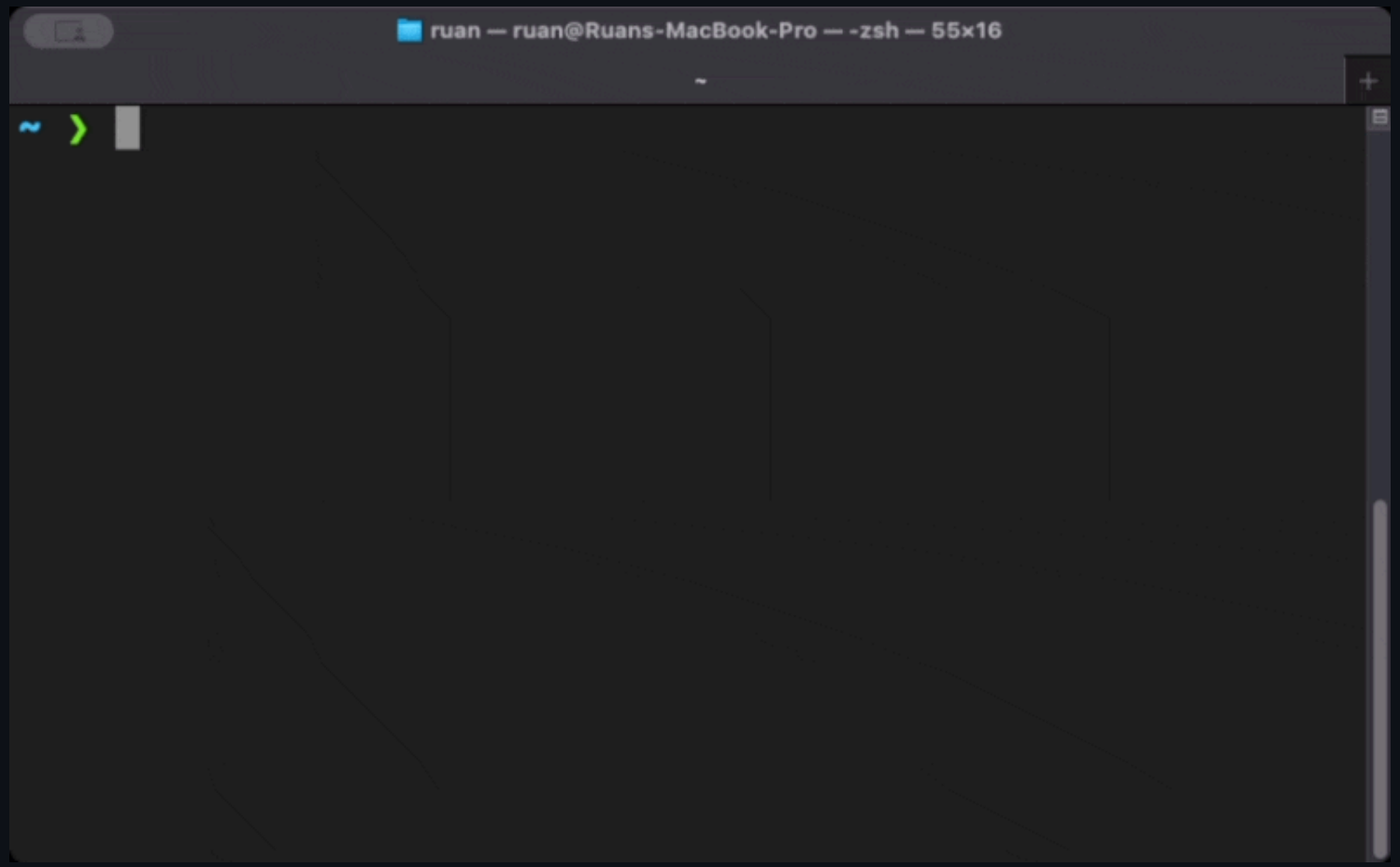
- ⬇️ Download app from `https://ollama.com/`
- Download LLM of choice

```
ollama pull llama3.1:8b
```

- To test, run LLM in terminal

```
ollama run llama3.1:8b
```

🦙 Testing Ollama in a Terminal



📊 Monitoring LLM Apps with Langfuse

- **What is Langfuse?**
  - Open Source LLM engineering platform
  - For tracing, evaluation, prompt management, etc.
  - Can be used to debug and improve your LLM apps
  - Can use as service or self-host

❓ Traces in Langfuse

> Intro to Observability & Traces in Langfuse

- `Trace`: Represents single request/operation of you app (with overall input and output)
- `Observation`: Each Trace can contain multiple observations that represent sub-seps
  - `Span`: General work step of some duration
  - `Generation`: Special spans that represent AI generation steps (contain extra metadata like token usage, model, etc.)

❓ Traces in Langfuse

> Example: RAG Trace in Langfuse

Preview | Scores

TRACE **RetrievalQA**
25/09/2024, 15:25:11
16.36s

Annotate | Add to dataset

Input

```
{
  query: "Explain how the different types of agent memory
  work"
}
```

Output

```
{
  result: "Based on the context provided, here's an
  explanation of how the different types of agent memory
  work: 1. **Memory Stream**: This is a long-term memory
  module that records a comprehensive list of agents'
  experiences in natural language. It appears to be an
  external database that stores all the interactions and
  experiences of the generative agents. The Memory Stream
```

TRACE RetrievalQA
16.36s

- SPAN RetrievalQA — 16.36s
  - SPAN VectorStoreRetriever — 0.38s
  - SPAN StuffDocumentsChain — 15.98s
    - SPAN LLMChain — 15.98s
      - GENERATION ChatOllama — 15.98s

🏗️ Setting Up Langfuse

> Option 1: Use as service

- Sign up at `https://cloud.langfuse.com/`
- Select region for hosting (`EU` or `US`)
- Create a new Project
- Generate API keys for sending traces

🏗️ Setting Up Langfuse

> Option 2: Locally, with Docker compose

- Requires `docker` and `docker compose` - get with Docker Desktop
- Run Docker compose to spin up local Langfuse

```
# Clone the Langfuse repository
git clone https://github.com/langfuse/langfuse.git
cd langfuse

# Start the server and database
docker compose up
```

🏗️ Setting Up Langfuse

> Before using in code

- Finally, pip-install the `langfuse` package

```
pip install langfuse
```

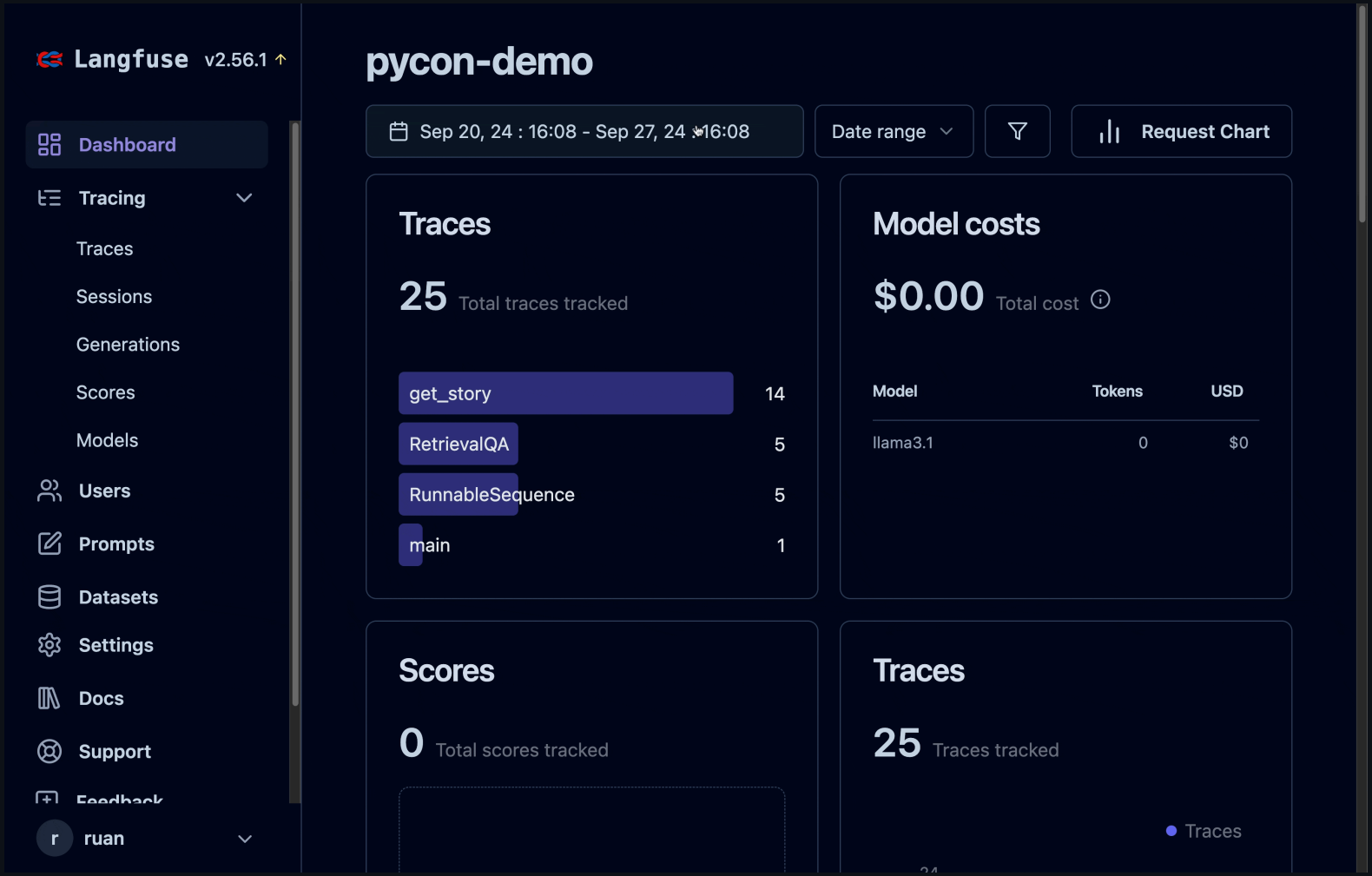
- And set these environment variables to communicate with your Langfuse instance

```
export LANGFUSE_SECRET_KEY="sk-..."
export LANGFUSE_PUBLIC_KEY="pk-..."
export LANGFUSE_HOST="https://cloud.langfuse.com" # or local instance
```

✅ Done Setting Up Langfuse

- Now we have our infrastructure set up
  - Langfuse server with web UI (at `localhost:3000`)
  - Postgres DB as backend (at `localhost:5432`)
- The Python package needed to communicate with it
- And we've pointed it to our instance

The Langfuse Dashboard



🎯 Monitoring with Langfuse

> Instrumenting your code

- Configure your app to talk to your Langfuse instance
  - Configure Langfuse to send traces to correct instance
  - Python decorator
  - LangChain callback handler

📡 Langfuse Instrumentation

> Python decorator (for any Python function)

```python
from langfuse.decorators import observe

@observe()
def call_llm(prompt: str):
    # Any code
    response: str = llm.invoke(prompt)
    return response

@observe()
def get_story():
    story = call_llm("Tell me a story")
    return story

get_story()
```

🔗 Langfuse Trace

> Traces, spans, nesting...

Preview | Scores

TRACE **get_story**
25/09/2024, 15:25:06

Annotate | + Add to dataset

Input

```
{
  args: [
  ]
  kwargs: {
  }
}
```

Output

```
"In a forgotten library, a young girl opened a dusty book and
discovered it was writing her life story as she lived it,
each turn of the page revealing her next decision."
```

TRACE get_story
- SPAN call_llm

📡 Langfuse Instrumentation

> LangChain callback handler (for automatic LangChain integration)

```
from langfuse.callback import CallbackHandler

langfuse_handler = CallbackHandler()

# Any LangChain Runnable (e.g. RAG chain)
rag_chain.invoke(
    "Explain how the different types of agent memory work",
    config={"callbacks": [langfuse_handler]}
)
```

🔗 Langfuse Trace

> Automatically labelled Traces/Spans

Preview | Scores

TRACE **RetrievalQA**
25/09/2024, 15:25:11
16.36s

Annotate | Add to dataset

Input

```
{
  query: "Explain how the different types of agent memory
  work"
}
```

Output

```
{
  result: "Based on the context provided, here's an
  explanation of how the different types of agent memory
  work: 1. **Memory Stream**: This is a long-term memory
  module that records a comprehensive list of agents'
  experiences in natural language. It appears to be an
  external database that stores all the interactions and
  experiences of the generative agents. The Memory Stream
```

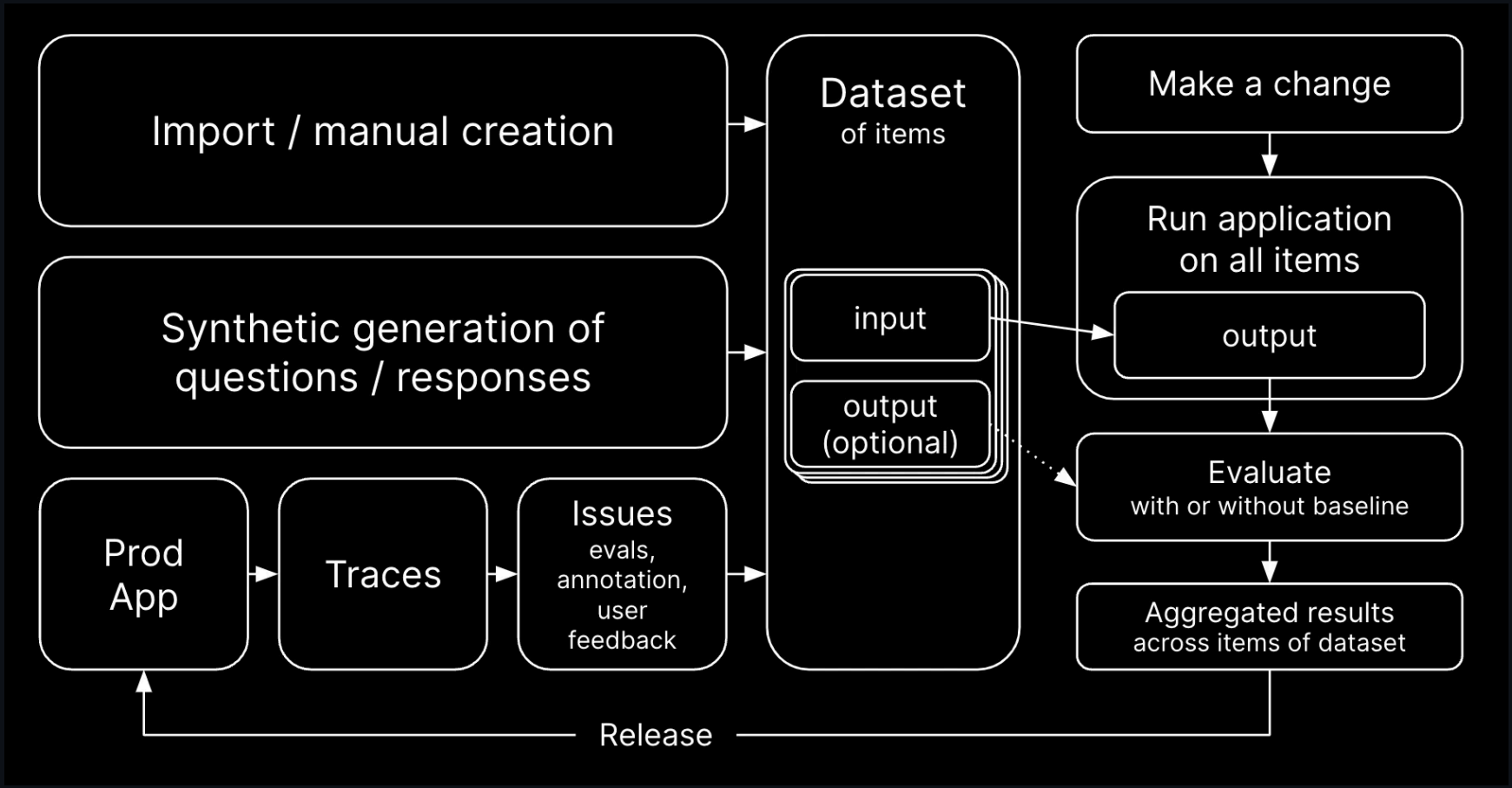
TRACE RetrievalQA
16.36s

- SPAN RetrievalQA — 16.36s
  - SPAN VectorStoreRetriever — 0.38s
  - SPAN StuffDocumentsChain — 15.98s
    - SPAN LLMChain — 15.98s
      - GENERATION ChatOllama — 15.98s

📊 LLM-Assisted Evaluation with Langfuse

- **Evaluation Datasets** 📚
  - To capture a set of inputs and expected outputs for your system
  - To log runs of your system versions on the data
  - For tracking performance over time or testing before pushing to prod
- **LLM-Assisted Scoring** 🎯
  - Attach a score to a trace (human or programmatic evaluation)
  - Automated evaluation using predefined metrics
  - Use as feedback loop to improve your system's performance

Langfuse Eval Flow



From https://langfuse.com/docs/datasets/overview

🧬 Langfuse Dataset

- First, create a Langfuse client with the Python SDK

```
from langfuse import Langfuse
langfuse = Langfuse()
```

- Then, create a dataset with a name

```
langfuse.create_dataset(name="eval-dataset-v1")
```

🔗 Langfuse Dataset

- Finally, populate the dataset with input-output pairs

```
dataset = {
  "question1": "answer1",
  ...
}

for question, expected_ans in dataset.items():
  langfuse.create_dataset_item(
    dataset_name="eval-dataset-v1",
    input=question,
    expected_output=expected_ans,
  )
```

Langfuse Dataset

Langfuse v2.56.1

- Dashboard
- Tracing
  - Traces
  - Sessions
  - Generations
  - Scores
  - Models
- Users
- Prompts
- Datasets
- Settings

pycon-demo

Datasets

(3/6) | New dataset

Name	Items	Created	Metadata
eval-dataset-v1	3	29/09/2024, 17:28:45	

🔗 Langfuse Dataset Runs

- Now you can pull the dataset

```
dataset = langfuse.get_dataset("eval-dataset-v1")
```

- And loop through items in the dataset for passing to your app

🧬 Add Loop Through Dataset and Eval Responses

```
for item in dataset.items:
  # Invoke app
  output, lf_trace_id = invoke_agent(question=item.input)

  # Link trace to dataset run
  item.link(
    run_name="Eval agent v0.0.1",
    run_metadata={...},
    trace_id=lf_trace_id,
  )

  # Eval
  c = get_correctness(output, item.expected_output)

  # Add score
  langfuse.score(
    trace_id=lf_trace_id,
    name="Answer Correctness,
    value=c
  )
```

🔗 Langfuse Trace Annotation

- As seen, you can attach values/scores to Traces
  - Using Python SDK (programmatic eval with `langfuse.score()`)
  - Or in the Dashboard (human eval, with `Annotate` button)
- These scores are aggregated per Dataset Run and shown in the Dashboard

🧬 Langfuse Trace Annotation

> Very useful to compare performance of your app versions

pycon-demo > Datasets > eval-dataset-v1

eval-dataset-v1

(4/7) | Runs | Items

Name	Description	Run Items	Scores (avg)	
Chatbot v0.0.2	Eval agent v0.0.2 on eval-dataset-v1	2	answer_correctness 0.58	context_recall 0.75
Chatbot v0.0.1	Eval agent v0.0.1 on eval-dataset-v1	3	answer_correctness 0.26	context_recall 0.00

💡 Tips and Caveats

LLM-assisted eval

- Can help, but it's not a silver bullet
- Use to inform priors. Look at the "direction" it's pointing you in, rather than absolute values
- Consider alternatives depending on your problem (e.g. execution evaluation for code generation)
- If used, consider Chain of Thought to improve results and have humans evaluate your LLM evaluations

🧬 Langfuse Conclusion

- This was just a quick overview of Langfuse
  - It also has other features, like prompt management
  - New updates released often
- More reasons to choose Langfuse:
  - Open-source
  - Low performance overhead
  - Multi-Modal tracing support
  - Public API for custom integrations
- Some Langfuse alternatives: Arize Phoenix, LangSmith (from LangChain)

🎉 Takeaways

- Consider **Ollama** for zero-cost local **prototyping and learning**
- Use **Langfuse** to help you with **monitoring and evaluation**
- Set up a **feedback** loop to keep **refining iterations** of your LLM apps
- Happy experimenting! 🧪

References

- https://langfuse.com/docs
- https://langfuse.com/guides/cookbook/datasets
- https://langfuse.com/guides/cookbook/evaluation_of_rag_with_ragas
- https://ollama.com/
- https://python.langchain.com/docs
- https://applied-llms.org/#evaluation-monitoring
- https://www.anthropic.com/news/evaluating-ai-systems

Thank you!

- GitHub: *@ruankie*
- Email: *ruan@melio.ai*
- Melio website: *melio.ai*

melio.ai