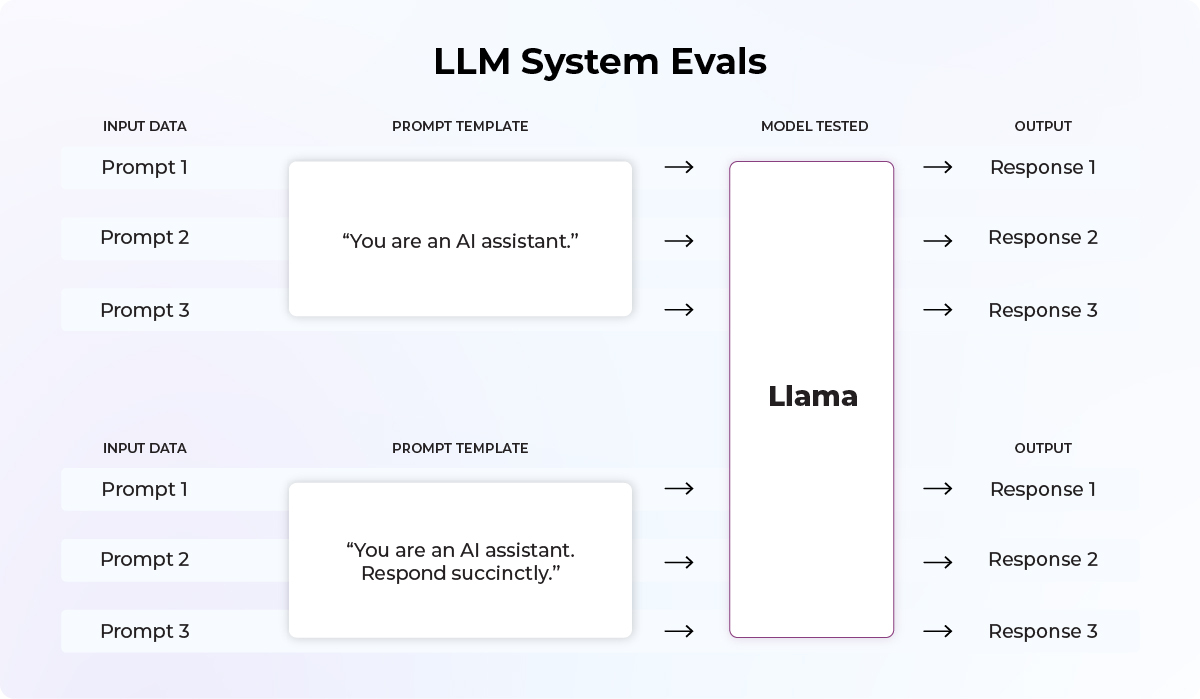
What Are LLM System Evaluations (Evals)?

**LLM system evaluation** is the complete evaluation of components that you have control of in your system. The most important of these components are the prompt (or prompt template) and context. LLM system evals assess how well your inputs can determine your outputs.

LLM system evaluation may, for example, hold the LLM constant and change the prompt template. Since prompts are more dynamic parts of your system, this evaluation makes a lot of sense throughout the lifetime of the project. For example, an LLM can evaluate your chatbot responses for usefulness or politeness, and the same eval can give you information about performance changes over time in production.

In this case, we are evaluating two different prompt templates on a single foundational model. We are testing the same dataset across the two templates and seeing how their metrics like precision and recall stack up.

When To Use LLM System Evaluations versus LLM Model Evaluations: It Depends On Your Role

There are distinct personas who make use of LLM evaluations. One is the model developer or an engineer tasked with fine-tuning the core LLM, and the other is the practitioner assembling the user-facing system.

There are very few LLM model developers, and they tend to work for places like OpenAI, Anthropic, Google, Meta, and elsewhere. **Model developers care about LLM model evals**, as their job is to deliver a model that caters to a wide variety of use cases.

For ML practitioners, the task also starts with model evaluation. One of the first steps in developing an LLM system is picking a model (i.e. GPT 3.5 vs 4 vs Palm, etc.). The LLM model eval for this group, however, is often a one-time step. Once the question of which model performs best in your use case is settled, the majority of the rest of the application’s lifecycle will be defined by LLM system evals. Thus, **ML practitioners care about both LLM model evals and LLM system evals but likely spend much more time on the latter.**

LLM System Evaluation Metrics

Having worked with other ML systems, your first question is likely this: “What should the outcome metric be?” The answer depends on what you are trying to evaluate.

* **Extracting structured information**: You can look at how well the LLM extracts information. For example, you can look at completeness (is there information in the input that is not in the output?).
* **Question answering**: How well does the system answer the user’s question? You can look at the accuracy, politeness, or brevity of the answer—or all of the above.
* **Retrieval Augmented Generation (**[**RAG**](https://arize.com/blog-course/introduction-to-retrieval-augmented-generation/)**)**: Are the retrieved documents and final answer relevant?

As a system designer, you are ultimately responsible for system performance, and so it is up to you to understand which aspects of the system need to be evaluated. For example, If you have an LLM interacting with children, like a tutoring app, you would want to make sure that the responses are age-appropriate and are not toxic.

What Are the Top LLM System Evaluation Metrics?

The most common evaluations we see being employed today are relevance, hallucinations, question-answering accuracy, and toxicity. Each one of these evals will have different templates based on what you are trying to evaluate. A fuller list of LLM system evaluation metrics appear below.

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Example Metrics** |
| Diversity | Examines the versatility of foundation models in responding to different types of queries | Fluency, Perplexity, ROUGE scores |
| User Feedback | Goes beyond accuracy to look at response quality in terms of coherence and usefulness | Coherence, Quality, Relevance |
| Ground Truth-Based Metrics | Compares a RAG system’s responses to a set of predefined, correct answers | Accuracy, [F1 score](https://arize.com/blog-course/f1-score/), Precision, Recall |
| Answer Relevance | How relevant the LLM’s response is to a given user’s query. | Binary classification (Relevant/Irrelevant) |
| [**QA Correctness**](https://docs.arize.com/phoenix/llm-evals/running-pre-tested-evals/q-and-a-on-retrieved-data) | Based on retrieved data, is an answer to a question correct? | Binary classification (Correct/Incorrect) |
| [**Hallucinations**](https://docs.arize.com/phoenix/llm-evals/running-pre-tested-evals/hallucinations) | Looking at LLM hallucinations with regard  to retrieved context | Binary classification (Factual/Hallucinated) |
| [**Toxicity**](https://docs.arize.com/phoenix/llm-evals/running-pre-tested-evals/toxicity) | Are responses racist, biased, or toxic? | Disparity Analysis, [Fairness Scoring](https://arize.com/blog-course/algorithmic-bias-examples-tools/), Binary classification (Non-Toxic/Toxic) |

Retrieval Metrics

In addition to these response metrics, in the case of retrieval augmented generation several other retrieval metrics can be helpful.

* **Contextual relevance** looks at relevance of the retrieved context to the original query. This can be a binary classification of relevant/irrelevant or ranking metrics can be used (i.e. [MRR](https://arize.com/blog-course/monitoring-collaborative-filtering-recsys/#metrics), Precision@K, [MAP](https://arize.com/blog-course/monitoring-collaborative-filtering-recsys/#metrics), [NDCG](https://arize.com/blog-course/ndcg/), etc.)
* **Faithfulness** or groundedness looks at how much the foundation model’s response aligns with retrieved context. For example, this can be a binary classification of faithful or unfaithful.

Concrete Example: Evaluating Relevance of Context

Here is an example for evaluating relevance of context using the open-source [Arize Phoenix](https://phoenix.arize.com/) tool for simplicity. Within the Phoenix tool, there exist default templates for most common use cases. Here is the one we will use for this example:

|  |
| --- |
| Relevance Eval Template: You are comparing a reference text to a question and trying to determine if the reference text contains information relevant to answering the question. Here is the data: [BEGIN DATA] \*\*\*\*\*\*\*\*\*\*\*\* [Question]: {query} \*\*\*\*\*\*\*\*\*\*\*\* [Reference text]: {reference} [END DATA]Compare the Question above to the Reference text. You must determine whether the Reference text contains information that can answer the Question. Please focus on whether the very specific question can be answered by the information in the Reference text. Your response must be single word, either “relevant” or “irrelevant”, and should not contain any text or characters aside from that word. “irrelevant” means that the reference text does not contain an answer to the Question. “relevant” means the reference text contains an answer to the Question. |

First, we will import all necessary dependencies:

from phoenix.experimental.evals import (

RAG\_RELEVANCY\_PROMPT\_TEMPLATE\_STR,

RAG\_RELEVANCY\_PROMPT\_RAILS\_MAP,

OpenAIModel,

download\_benchmark\_dataset,

llm\_eval\_binary,

)

from sklearn.metrics import precision\_recall\_fscore\_support

Now, let’s bring in the dataset:

# Download a "golden dataset" built into Phoenix

benchmark\_dataset = download\_benchmark\_dataset(

task="binary-relevance-classification", dataset\_name="wiki\_qa-train"

)

# For the sake of speed, we'll just sample 100 examples in a repeatable way

benchmark\_dataset = benchmark\_dataset.sample(100, random\_state=2023)

benchmark\_dataset = benchmark\_dataset.rename(

columns={

"query\_text": "query",

"document\_text": "reference",

},

)

# Match the label between our dataset and what the eval will generate

y\_true = benchmark\_dataset["relevant"].map({True: "relevant", False: "irrelevant"})

Now let’s conduct our evaluation:

# Any general purpose LLM should work here, but it is best practice to keep the temperature at 0

model = OpenAIModel(

model\_name="gpt-4",

temperature=0.0,

)

# Rails will define our output classes

rails = list(RAG\_RELEVANCY\_PROMPT\_RAILS\_MAP.values())

benchmark\_dataset["eval\_relevance"] = \

llm\_eval\_binary(benchmark\_dataset,

model,

RAG\_RELEVANCY\_PROMPT\_TEMPLATE\_STR,

rails)

y\_pred = benchmark\_dataset["eval\_relevance"]

# Calculate evaluation metrics

precision, recall, f1, support = precision\_recall\_fscore\_support(y\_true, y\_pred)

How To Get AI To Evaluate AI

There are two distinct steps to the process of evaluating your LLM-based system with an LLM. First, establish a benchmark for your LLM evaluation metric. To do this, you put together a dedicated LLM-based eval whose only task is to label data as effectively as a human labeled your “golden dataset.” You then benchmark your metric against that eval. Then, run this LLM evaluation metric against results of your LLM application (more on this below).

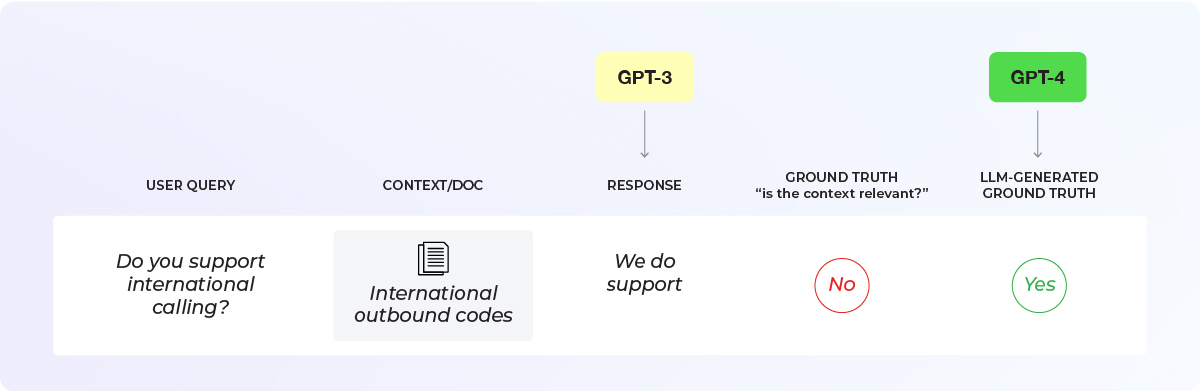
How To Build An LLM Evaluation

The first step, as we covered above, is to build a benchmark for your evaluations.

To do that, you must begin with a **metric best suited for your use case**. Then, you need the **golden dataset**. This should be representative of the type of data you expect the LLM eval to see. The golden dataset should have the “ground truth” label so that we can measure performance of the LLM eval template. Often such labels come from human feedback. Building such a dataset is laborious, but you can often find a standardized one for the most common use cases (as we did in the code above).



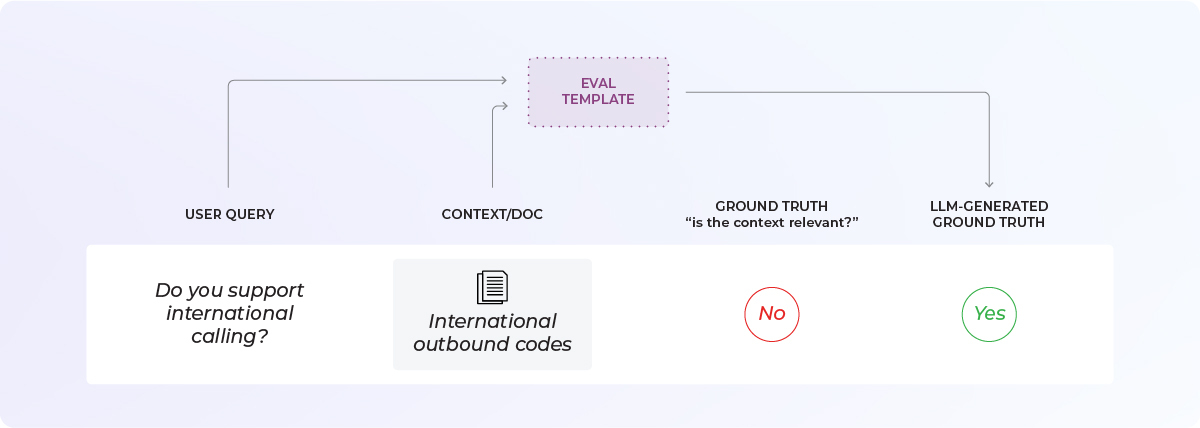
Then you need to decide **which LLM** you want to use for evaluation. This could be a different LLM from the one you are using for your application. For example, you may be using Llama for your application and GPT-4 for your eval. Often this choice is influenced by questions of cost and accuracy.



Now comes the core component that we are trying to benchmark and improve: the **eval template**. If you’re using an existing library like OpenAI or Phoenix, you should start with an existing template and see how that prompt performs.

If there is a specific nuance you want to incorporate, adjust the template accordingly or build your own from scratch. Keep in mind that the template should have a clear structure. Be explicit about the following:

1. **What is the input?** In our example, it is the documents/context that was retrieved and the query from the user.
2. **What are we asking?** In our example, we’re asking the LLM to tell us if the document was relevant to the query
3. **What are the possible output formats?** In our example, it is binary relevant/irrelevant, but it can also be multi-class (e.g., fully relevant, partially relevant, not relevant).

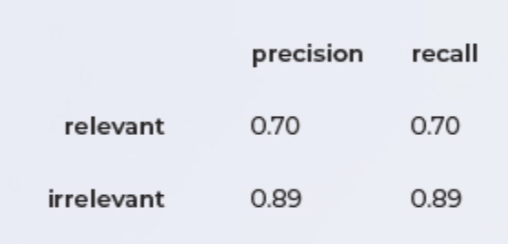


You now need to run the eval across your golden dataset. Then you can **generate metrics** (overall accuracy, [precision, recall](https://arize.com/blog-course/precision-vs-recall/), [F1-score](https://arize.com/blog-course/f1-score/), etc.) to determine the benchmark. It is important to look at more than just overall accuracy. We’ll discuss that below in more detail.

If you are not satisfied with the performance of your LLM evaluation template, you need to change the prompt to make it perform better. This is an iterative process informed by hard metrics. As is always the case, it is important to avoid overfitting the template to the golden dataset. Make sure to have a representative holdout set or run a [k-fold cross-validation](https://arize.com/blog/cross-validation-machine-learning/).



**Finally, you arrive at your benchmark**. The optimized performance on the golden dataset represents how confident you can be on your LLM eval. It will not be as accurate as your ground truth, but it will be accurate enough, and it will cost much less than having a human labeler in the loop on every example.



Why Is It Important To Use Precision and Recall When Benchmarking An LLM Prompt Template?

The industry has not fully standardized best practices on LLM evals. Teams commonly do not know how to establish the right benchmark metrics.

Overall accuracy is used often, but it is not enough.

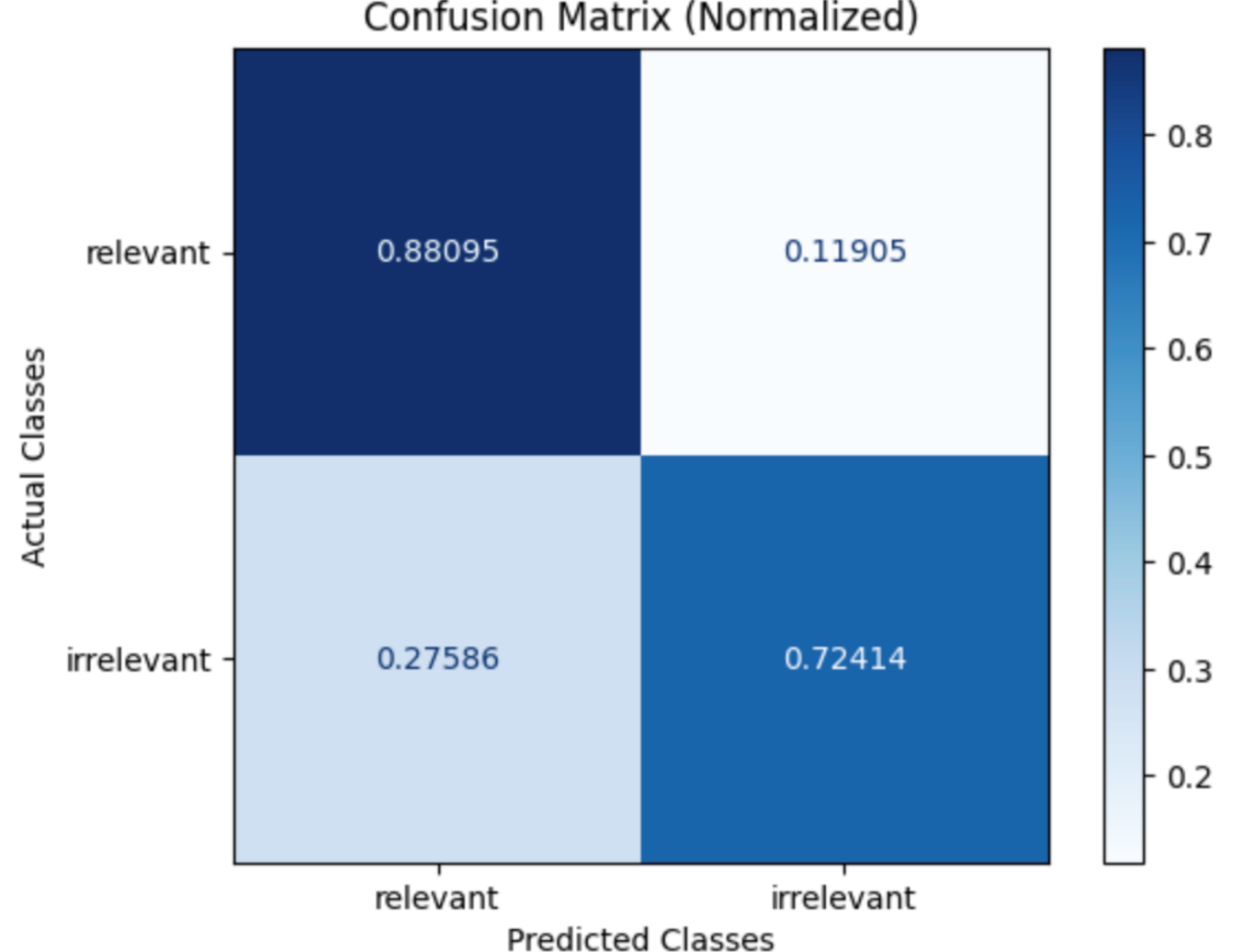
This is one of the most common problems in data science in action: very significant class imbalance makes accuracy an impractical metric.

Thinking about it in terms of the relevance metric is helpful. Say you go through all the trouble and expense of putting together the most relevant chatbot you can. You pick an LLM and a template that are right for the use case. This should mean that significantly more of your examples should be evaluated as “relevant.” Let’s pick an extreme number to illustrate the point: 99.99% of all queries return relevant results. Hooray!

Now look at it from the point of view of the LLM eval template. If the output was “relevant” in all cases, without even looking at the data, it would be right 99.99% of the time. But it would simultaneously miss all of the (arguably most) important cases — ones where the model returns irrelevant results, which are the very ones we must catch.

In this example, accuracy would be high, but precision and recall (or a combination of the two, like the F1 score) would be very low. Precision and recall are a better measure of your model’s performance here.

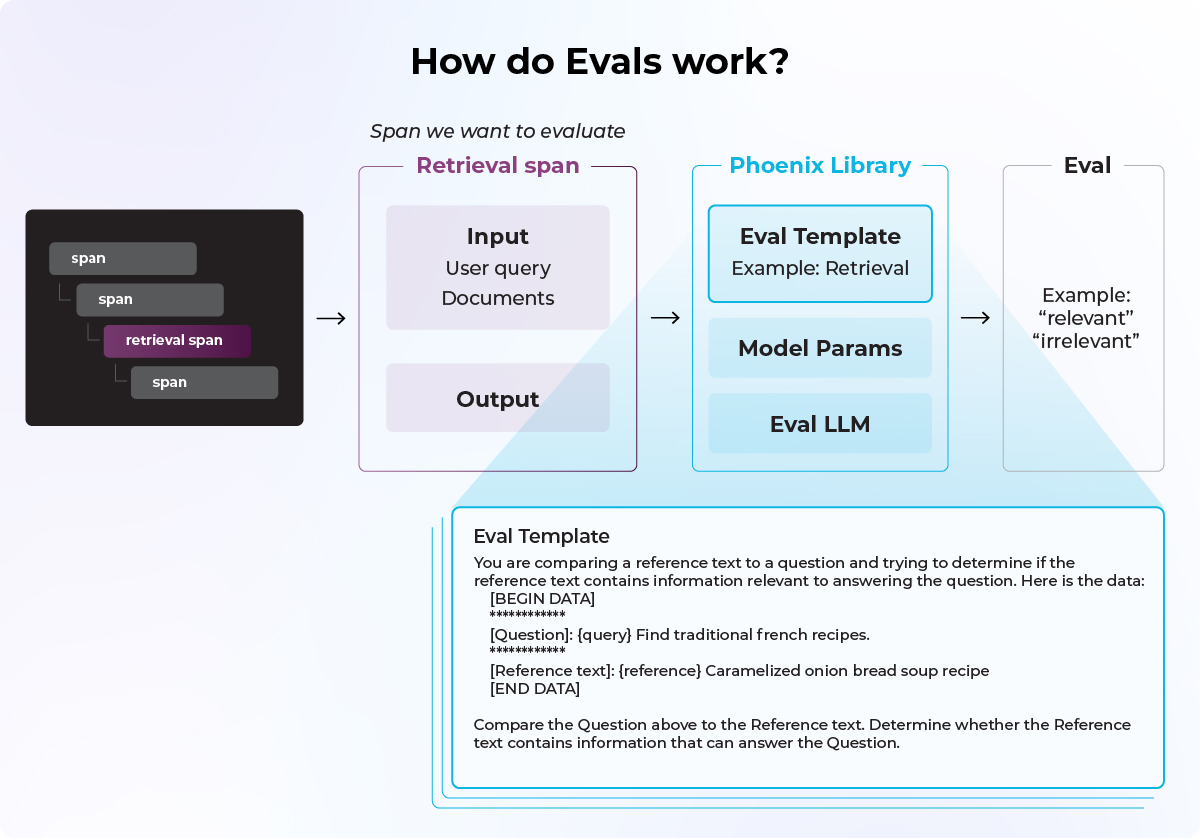
The other useful visualization is the confusion matrix, which basically lets you see correctly and incorrectly predicted percentages of relevant and irrelevant examples.

*In this example, we see that the highest percentage of predictions are correct: a relevant example in the golden dataset has an 88% chance of being labeled as such by our eval. However, we see that the eval performs significantly worse on “irrelevant” examples, mislabeling them more than 27% of the time.*

How To Run LLM Evals On Your Application

At this point you should have both your LLM application and your tested LLM eval. You have proven to yourself that the eval works and have a quantifiable understanding of its performance against a golden dataset.

Now we can actually use our eval on our application. This will help us measure how well our LLM application is doing and figure out how to improve it.



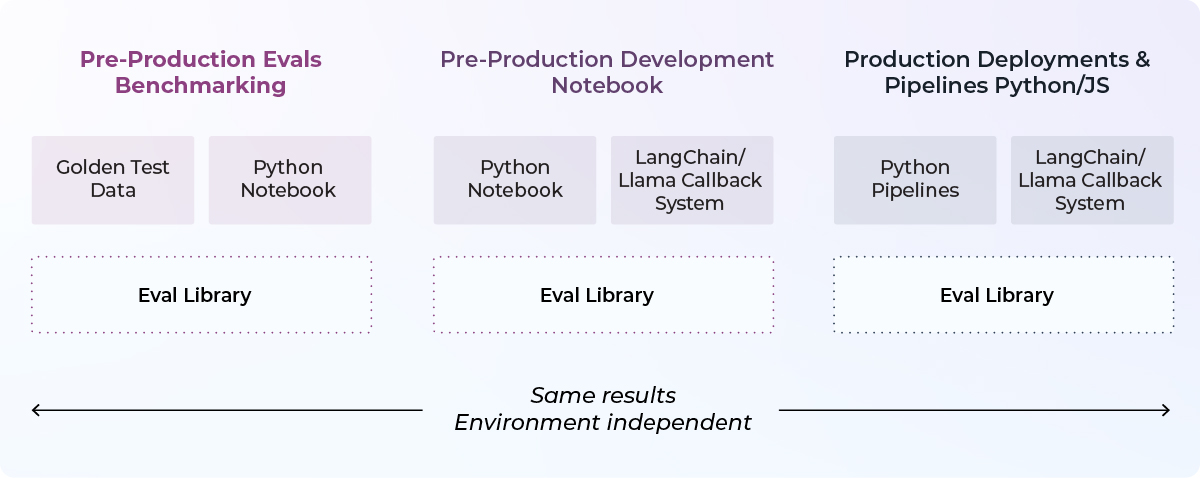
The LLM system eval runs your entire system with one extra step. For example:

1. You retrieve your input docs and add them to your prompt template, together with sample user input.
2. You provide that prompt to the LLM and receive the answer.
3. You provide the prompt and the answer to your eval, asking it if the answer is relevant to the prompt.

It is a best practice not to do LLM evals with one-off code but rather a library that has built-in prompt templates. This increases reproducibility and allows for more flexible evaluation where you can swap out different pieces.

These evals need to work in three different environments:

1. Pre-production when you’re doing the benchmarking. (
2. Pre-production when you’re testing your application. This is somewhat similar to the offline evaluation concept in traditional ML. The idea is to understand the performance of your system before you ship it to customers.
3. Production when it’s deployed. Life is messy. Data drifts, users drift, models drift, all in unpredictable ways. Just because your system worked well once doesn’t mean it will do so on Tuesday at 7 p.m. Evals help you continuously understand your system’s performance after deployment.



Questions To Consider When Building an LLM Evaluation Strategy

How many rows should you sample?

The LLM-evaluating-LLM paradigm is not magic. You cannot evaluate every example you have ever run across—that would be prohibitively expensive. However, you already have to sample data during human labeling, and having more automation only makes this easier and cheaper. So you can sample more rows than you would with human labeling.

Which evals should you use?

This depends largely on your use case. For search and retrieval, relevancy-type evals work best. Toxicity and hallucinations have specific eval patterns. There is a discussion of this in section 3.

Some of these evals are important in the troubleshooting flow. Question-answering accuracy might be a good overall metric, but if you dig into why this metric is underperforming in your system, you may discover it is because of bad retrieval, for example. There are often many possible reasons, and you might need multiple metrics to get to the bottom of it.

What model should I use?

It is impossible to say that one model works best for all cases. Instead, you should run model evaluations to understand which model is right for your application. You may also need to consider tradeoffs of recall vs. precision, depending on what makes sense for your application. In other words, do some data science to understand this for your particular case.

*Which model to use often depends on your task*

Conclusion

Being able to evaluate the performance of your application is very important when it comes to production code. In the era of LLMs, the problems have gotten harder, but luckily we can use the very technology of LLMs to help us in running evaluations. Such evaluation should test the whole system and not just the underlying LLM model—think about how much a prompt template matters to user experience. Best practices, standardized tooling, and curated datasets simplify the job of developing LLM systems.