LARGE LANGUAGE MODEL

SYSTEM EVALS

IN THE WILD

Issue 1

TABLE OF CONTENTS

LLM evaluations are a hot topic as companies move past the flashy AI demos. A common refrain from engineers and leaders is: "How can we trust these systems are doing the right thing?"

Evals keep coming up as the answer, the thing "you should" do. But it's not exactly a standard part of the software engineering toolkit.

So we pulled from Sri's seven years as a research engineer at Google, plus a ton of best practices from around the web. We decided to make a zine that could take you from zero to eval" in a way that's actually fun to read. When you create an eval, you're essentially defining what "good" looks like for your system.

You don't need a huge team or complex processes to do this. At Google, sure, some departments had entire teams dedicated to defining quality. But I've also seen plenty of solid evals done with just a spreadsheet and a Jupyter notebook.

So let's learn about evals!

Sri and Wil

EVAL SHOWS THE WAY

Sci-fi depictions of working with futuristic AI is like a dance between equal partners. But building current AI apps based on LLMs (Large Language Model) are more like working with a junior colleague that works really fast, but needs to be provided with a lot of context and details to work. In addition, its output can vary from run to run, even with the same input.

As of this publication, many AI engineers do vibe-check evals with the LGTM@K (Looks Good To Me) metric, often taking random stabs in the dark to fix and improve the system.

Unlike model evals, system evals help AI engineers improve apps with consistency. **This issue focuses exclusively on building system evals** to help steer consistent improvement in your AI apps.

Building AI apps without system evals is like driving without a map: you're never sure if you're getting closer. Even simple evals will show you the way better than nothing.

EVALS ARE FOR EVERYONE

It's work to set up a consistent and repeatable way of judging an LLM's output over time, but it pays dividends.

It's hard to describe exactly what "good" entails, even when using LLMs to detail it in natural language. It's hard to quantify qualitative metric we want in our output, such as relevance, groundedness, and non-toxicity.

With all these barriers, many AI engineers don't think they have time or the inclination for eval and that it's solely for the domain of large companies with the resources to implement it.

That's simply not true. There is always some level of eval which is right for you and your app at any stage of maturation. 👉

Different Levels of Eval

There are increasing levels of sophistication. Pick the right level for the size that you're at.

VIBES-BASED HUMAN EVAL

Using the LGTM (Looks good to me) metric, AI engineers eyeball the output. Without metrics to guide their priorities or work, there are times when it would have been better if they did nothing at all.

PROPERTY-BASED TESTS

Also called unit tests, these are aspects of the LLM output that can be checked by code, such as length or formatting. Often implemented as regexes or employing a linter to validate the output.

SYSTEMATIC HUMAN EVAL

There is now a process for evals, while still relying on human judgment. While the gold standard for quality, it's slow and not very scalable. But the examples gather here is the basis for automating eval.

LLM-AS-A-JUDGE EVAL

Using LLMs to judge LLM output sounds odd, but we have humans judging each others' work all the time. We can align an LLM to match human judgments to help scale up iteration towards better outputs.

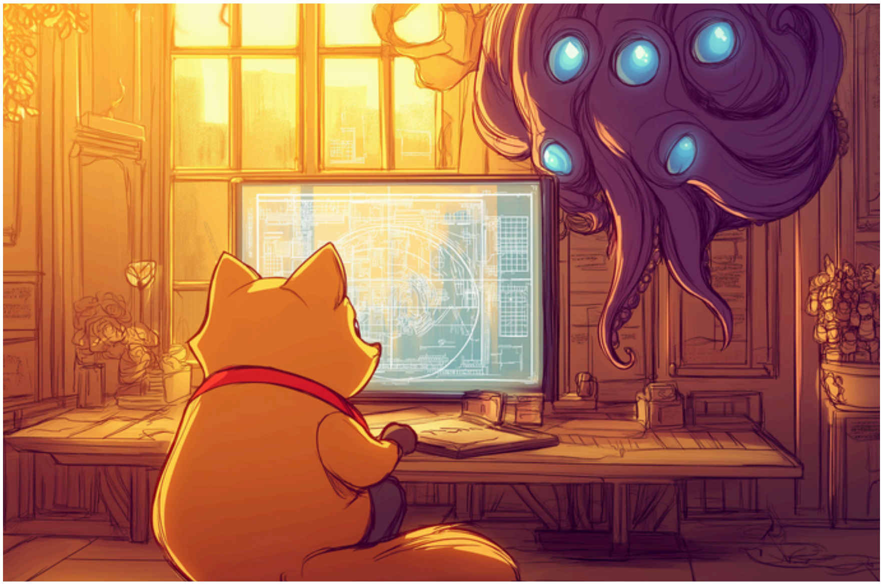
THE ROAD AHEAD

Even without full-blown eval, just having systematic ways of doing eval will yield data to help you prioritize where to focus your efforts on improving the system, to stop wasting effort, and to secure resources for building new capabilities.

What can we do to quantify qualitative metrics? While we can't write down a formula for "good", we can collect a corpus of examples of what is considered good.

From this golden dataset of examples, we can write down guidelines for human judges or a few-shot prompt for an LLM that can reflect the judgment shown by this dataset.

We'll show you how to build your own eval. We won't recommend specific tools, as those get obsoleted in this fast moving space. Instead, we'll equip you with principles and methodology to build your own eval, at whatever level of maturation of your current LLM application. 🍃



WHY DESIGN YOUR OWN EVAL?

Aren't LLMs already evaluated? Yes and no. The companies that trained the LLMs evaluate the models to measure their overall capabilities (**model evals**), but not their performance on specific tasks for real-world users (**system evals**).
This distinction is crucial because the real world is messy. Models can behave unexpectedly when exposed to real-world scenarios, which is why designing your own evaluation system is essential. Generic model evaluations simply can't capture the nuances of your specific use case.

Custom evaluations allow you to focus on your unique application and user needs. With your domain knowledge, you'll need to articulate to an LLM what constitutes "good", whether by principle or by examples. This approach provides insights that general evaluations miss, helping you understand how your system performs in the context that matters most - your own.

A well-designed evaluation system is systematic and metric-driven. This approach helps direct your efforts efficiently, following the 80/20 principle where 20% of your work yields 80% of the results. It organizes your thoughts on how to improve the system, reducing confusion and anxiety in the process. 👉

With clear metrics, you can prioritize your efforts and focus on the most impactful improvements.

Custom evals allow your team to ship with more confidence. Product managers and leaders can correlate eval metrics with business KPIs. Engineers can make more informed architectural decisions like which vector database to use, whether to fine-tune, or whether to use GPT-4 or Claude.

One of the key benefits of custom evaluations is their ability to identify edge cases. These rare but critical failure modes are often specific to your system and might go unnoticed in general evaluations. By uncovering these edge cases, you can address potential issues before they become problems in real-world applications.

Custom evaluations also allow you to track progress over time. As you iterate on your system, you can consistently measure improvements, giving you a clear picture of your development trajectory. This ongoing assessment helps you understand the impact of changes and guides future development decisions.

Finally, the ability to demonstrate measured improvements can be a powerful tool in securing resources for future projects. When you can show concrete progress and improvements backed by data, it becomes easier to justify investments in your work and related initiatives. 🍃

COMPLEX AND MESSY DATA

Many vertical-specific SaaS applications require cleaning multi-dimensional data from inconsistently formatted spreadsheets. Consumer-facing AI systems may have to parse user intents from unstructured, informally written text.

INTEGRATING WITH TOOLS

RAG (Retrieval Augmented Generation) systems might need to query internal data stores by formulating valid SQL queries or following a specific JSON/XML schema. Agent systems need to parse and transform the output of their tools to execute their plan.

DOMAIN-SPECIFIC REQUIREMENTS

In healthcare, your system may have to understand abbreviations and ontologies like CPT codes.

If you’re building in the legal space, it may have to comply with strict formatting and wording requirements.

A Typical LLM App Architecture

Before diving into evals, let's take stock of a typical LLM app architecture.

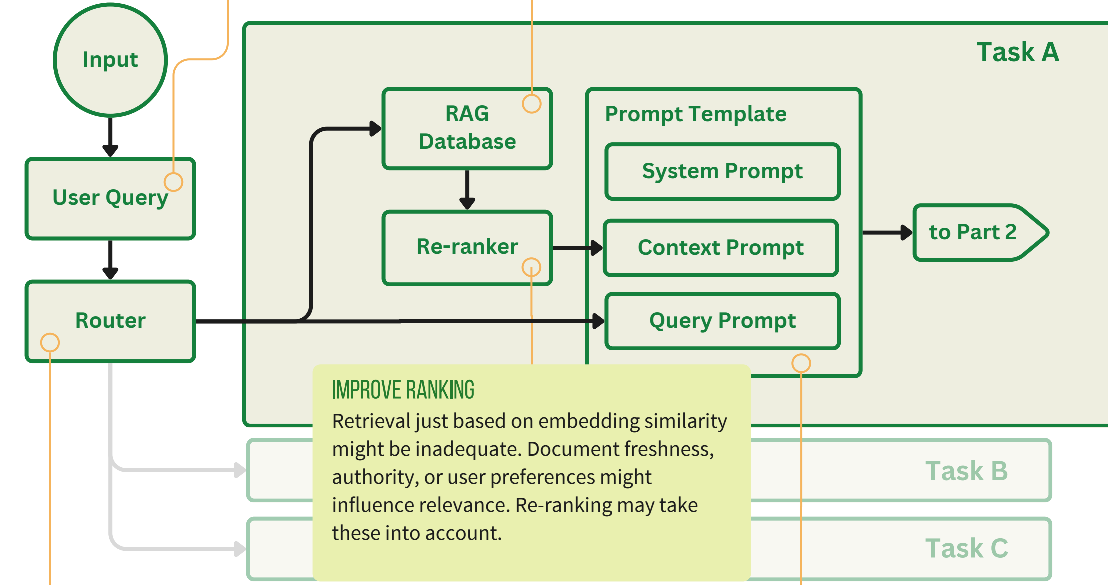
First, the user query is used to fetch relevant documents from a RAG database. With this set of documents, we can re-rank them to provide the most relevant context for the LLM. Then we can combine this context with the user query in a combined 👉

USER QUERY

User query is inserted into the prompt template. It can also be used in control flow to decide which template to use.

NOT ONLY DATABASES

RAG (Retrieval Augmented Generation) doesn't have to be a database. It can be an external API call or a search index. Anything that provides relevant context to a user's query will



IMPROVE RANKING

Retrieval just based on embedding similarity might be inadequate. Document freshness, authority, or user preferences might influence relevance. Re-ranking may take these into account.

RULES OR LLM-DRIVEN ROUTER

Routes the user query to the LLM task that best handles the query. Sometimes, we can reject a query completely if we know we don't perform well on that kind of query with existing LLM tasks.

GOOD PROMPTS HAVE DETAILS

This combines the overall task description with the context and the user query into a complete description of the task for the LLM to do the task well.

prompt to send to the LLM for a task output.

In simpler RAG (Retrieval Augmented Generation) apps, there is just a single LLM task to handle all queries. But more complex apps like a question-answer system might have multiple specialized tasks where queries are directed by a router.

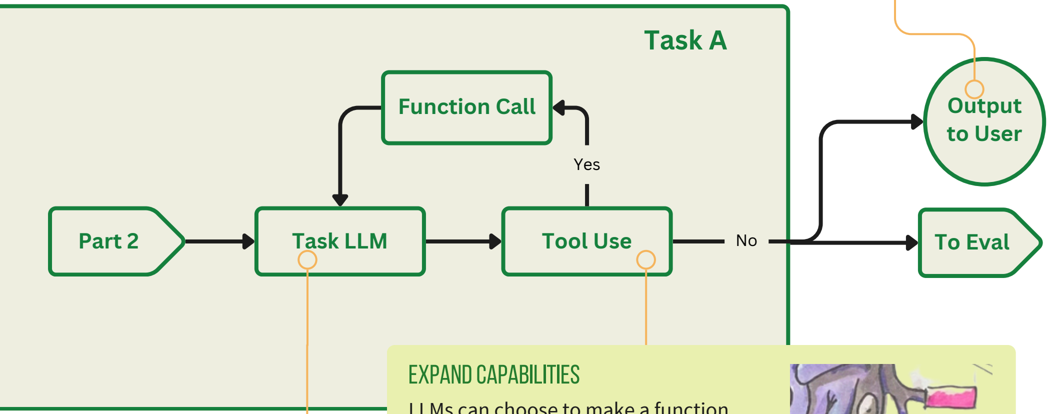
LLMs aren't great at certain types of tasks, such as logical reasoning and routine calculations.

But LLMs can leverage existing tools such as executing code or inference engine by using the LLM output to drive these tools.

Finally, the output is sent to the user and stored for eval. Let's see what an eval process looks like. 🍃

LOG FOR EVAL

For offline evals, the output is stored for later processing.



EXPAND CAPABILITIES

LLMs can choose to make a function call to generate output more accurate than LLMs could alone, such as doing arithmetic or logical reasoning.

CHOICE OF MODELS

There are many choices of models. Eval can help choose the model that balances latency, task performance, and cost.

Eval Feedback Loop Overview

System eval boils down to creating a systematic process for capturing the intuition for qualitative metrics and using it to judge subsequent LLM outputs.

These qualitative metrics, such as relevance, toxicity, hallucinations, or brand voice alignment are hard to express formulaically. To capture the intuition, we gather a corpus of example query/output/judgment triplets as our golden dataset. Then we iterate in a feedback loop.

First, test a human or LLM judge on these golden dataset triplets. Second, measure how well they match the ground truth judgments with metrics. Lastly, look at which examples they fail on, and tweak the human judge's guidelines or LLM eval prompt. Rinse and repeat.

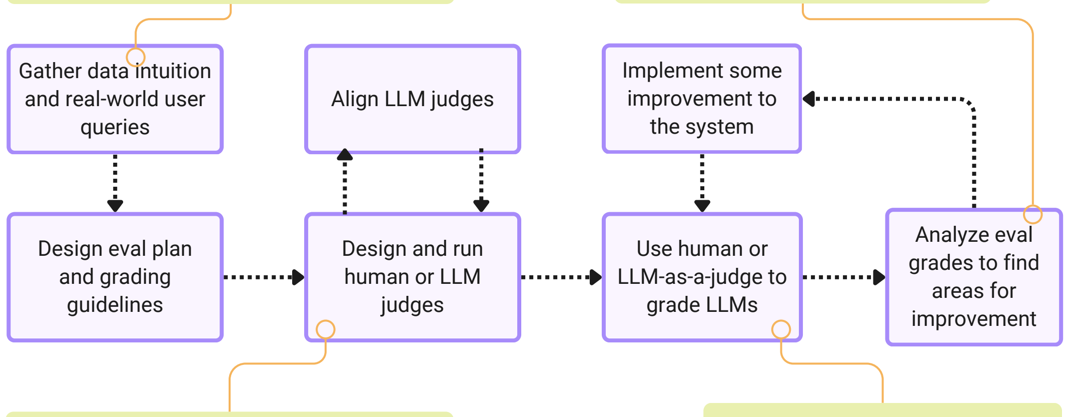
Once we have a judge that aligns well with the intuition for the qualitative metric, we can use it to judge production LLM outputs given user queries. 🍃

GATHER REAL-WORLD DATA

Collect human judgments on query/output pairs. You don't need that many to start--about 50 to 100. Gather than while doing vibe-check evals with your users. If you have no users, you can bootstrap user queries from your RAG documents.

ITERATE, ITERATE, ITERATE

Look at the individual examples and see which ones the judge missed. Based on analysis and talking to domain experts, hatch a hypothesis for what would improve the scores, then implement it.


Gather data intuition and real-world user queries
Align LLM judges
Implement some improvement to the system
Design eval plan and grading guidelines
Design and run human or LLM judges
Use human or LLM-as-a-judge to grade LLMs
Analyze eval grades to find areas for improvement

DESIGNING AND ALIGNING EVALS IS ITERATIVE

Not shown in this overview, but there's also an iteration loop here to align the LLM-as-a-judge with the intuitive notion of good outputs, as captured by a golden database of judgements on user query/output pairs.

SCORE JUDGES USING METRICS

We use precision/recall or other metrics to see how well our judge (human or LLM) matches the intuition captured by the golden dataset.

TYPES OF EVALS

System Eval vs Model Eval

To be clear, there are two different kinds of LLM evals: ***model evals*** and ***system evals*** (also called task or downstream evals).

Model evals rank different LLM models against each other based on their benchmark performance. While AI engineers use it occasionally to select models, it's mostly used by ML researchers to judge the quality of their models. relative to other models.

System evals measure how well your LLM app pipeline performs on a specific user tasks in the interest of improving performance over time. AI engineers will spend most of their eval time on this type of eval. 🍃

CATEGORY	SYSTEM EVAL	MODEL EVAL
SOURCE OF TRUTH	A golden dataset curated by domain experts. Expanded with synthetic data generated from LLMs	Based on benchmarks
NATURE OF QUESTIONS	Uses task specific questions to mimic real-world scenarios	Uses a set of standardized questions across a variety of scenarios
FREQUENCY OF EVAL	Used on every iteration to improve the response of an app	Once for every new version that comes out
PURPOSE	For refining output for deployed real-world applications	For improving general models over time
VALUE FOR EVALUATOR	Provides action items for improvements for task output	Provides apples to apples comparison between models
TYPE OF EVALUATOR	AI engineer building and supporting LLM application	Machine learning researcher or ML engineer building new models

Human judge or LLM-as-a-judge

System eval attempts to measure the quality of LLM outputs with qualitative metrics like relevance, toxicity, and groundedness.

These judgments can be made with a human judge or an LLM judge. Most likely, you'll end up using a mix of both. It'll be impossible to completely offload judgments to an LLM. There will be edge cases, queries drifting over time, and other things that will always require human judgment and intervention.

It's best to think of LLM judges as augmentations to scale human judgments, rather than a complete automation solution to eval. 🍃

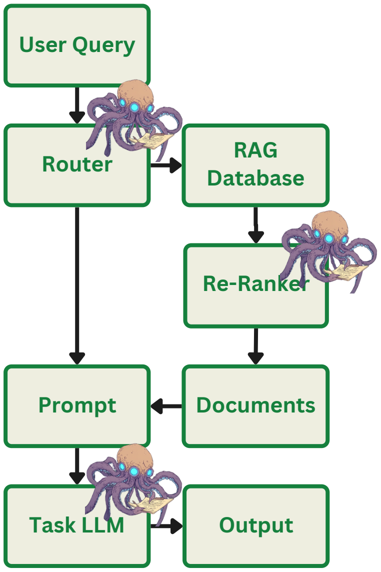
HUMAN JUDGE

Human (or friendly forest animal) judgment on the performance of our LLM task can be costly in time and money, but it's the gold standard.

LLM-AS-A-JUDGE

LLM judgement is cheaper and faster than humans. But they need to be tuned to match human level judgements.

Eval of LLM-driven Components



Even a moderately complicated LLM pipeline can have multiple LLM-driven components. While system evals typically judge end-to-end performance, eventually we'll want evals for individual LLM-driven components. These components can be interdependent, and just a single end-to-end eval may not be enough to know where to improve your system.

Major components like RAG (Retrieval Augmented Generation), the Query Router, and the Synthetic Data Generator can have their own prompts and hence their own evals. However, the optimization of a part should never take precedence over the optimization of the whole.

Let's see what happens when no evals are in place at the Forest Cafe. 🍃