What To Ask Your Engineers

Evaluating AI for Less-Technical Stakeholders

ABIGAIL HADDAD

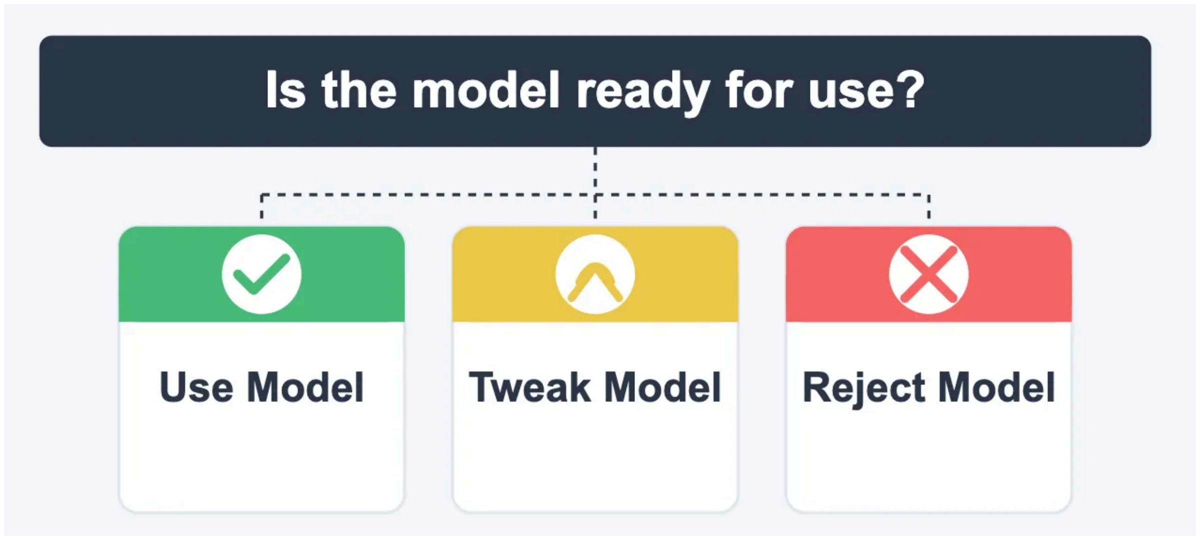
APR 23, 2025

3

Share

When I'm prototyping a model, my goal is to share information that's useful to decision-makers. That means figuring out not just what *I* know, but what *they* need know—and how to present it in a way that works for them. Without clear direction, though, I'm often guessing. I know the model better than anyone—that's my job—b I'm not the expert in the real-world problem we're trying to solve with it.

That dynamic is typical: your data scientist or engineer hopefully knows the model inside and out, but they may not understand the broader context or constraints you working within.

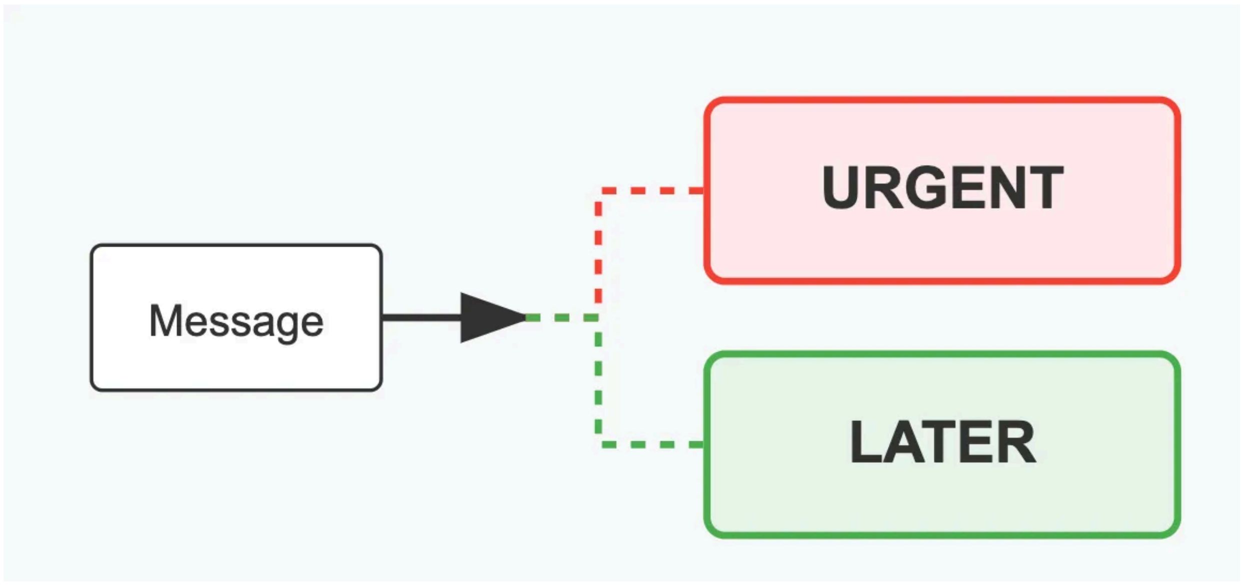


This is about how to have a conversation with that person and what to ask for so th you can be better-informed when deciding whether this model is good enough to u: whether it needs to be tweaked, or whether it's just not going to work unless it can significantly improved.

The Example: Emergency Communications Model

When I'm talking about a model, I mean anything from a linear regression model tc chatbot. But for this post, let's imagine I've built a text classification model that processes messages during disasters to flag messages requiring immediate respons Let's say we're getting these messages from various sources, constantly, and we wa to see if we can automate classifying them as either “urgent" or "we can look at this later."

The communications flagged as "urgent" get immediate human attention, while the others will be looked at when someone has time for them.



Currently, your organization is doing all of this by hand, which means someone is spending a lot of time sorting these messages instead of actually responding! We're exploring whether we can create a model that's good enough to replace some or all this triage process.

Your engineer wants to get you the information you need to make a decision. How (you figure out what to ask for?

Here are the strategies I recommend to help you answer that question.

Strategy 1: Think About Risk

When considering risk, ask yourself: what keeps you up at night?

In this case, getting some of these right could actually be the difference between lif and death. But when that's not the case, I bet there are still outcomes you're especi concerned about. What could go wrong that would lose your organization a lot of money, or where your leadership would be getting pulled away from their normal w to answer for your model, or where someone could successfully sue you?

In this specific context, risks could include:

- The model misses too many urgent messages overall
- It sometimes misses a certain really urgent category which we can't afford to *e* miss
- It systematically performs poorly on certain types of messages, like those in otl languages
- It's inconsistent (or "non-deterministic") and gives different outputs for the sar input, and we need that not to happen.

Your engineer will have ideas, but ultimately you are the one who is going to have t figure out what those keep-you-up-at-night scenarios are.

The reality is that no model performs perfectly–and if someone says it does, if you'r ever looking at something that's claiming 99.9% accuracy, be skeptical. So the quest isn't "can we build a perfect model?" but rather "which errors can we tolerate, and which ones would be catastrophic?"

Strategy 2: Get Both Broad AND Specific

Now that we've identified the risks we're concerned about, we're going to take then into the second strategy: get broad *and* specific.

This means translating your concerns into metrics we can track.

Broad metrics you should almost always ask for when you have this type of model— and which your engineer will probably automatically give you—include:

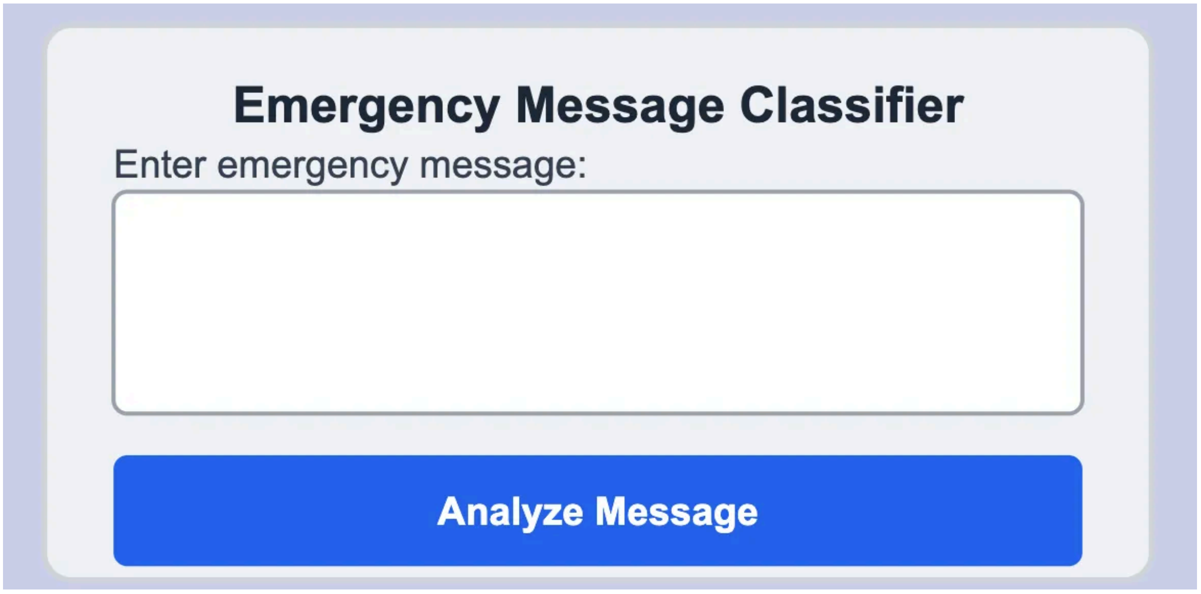
- **Precision**: If we flag something as urgent, what's the chance it's actually urgen (Or: how many false alarms will I get?)
- **Recall**: What percentage of truly urgent things are we catching? (Or: how many emergencies are we missing?)

There may be other important 'broad' ones depending on your risks. For instance, a model that performs well overall but terribly on non-English messages might not b acceptable in your context, so you might also track performance by language. Depending on the results, you might test out using machine translation prior to sending messages to your model, or if that doesn't work well, always have a person read through the non-English ones soon after they're received.

Include specific test cases that are important to you and where you know what the expected output should be. For thoroughness, include test cases that weren't used b the engineering team during development or testing.

Specific test cases can uncover problems that overall statistics might miss—and tha your engineers might not know are particularly important.

You can either test these yourself if you have access to the model or ask your engin to test and report back.



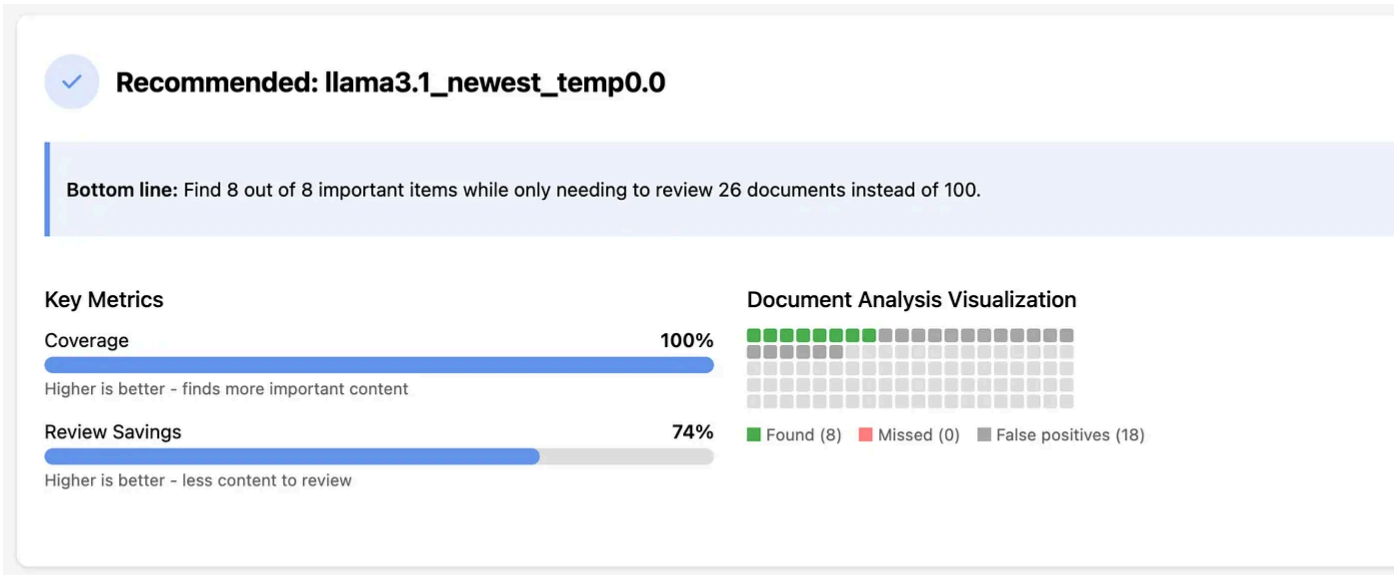
Your engineers can create a test interface for you to try specific examples

Strategy 3: Ask for Automation

Finally, once you know your risks and you've got the metrics, you'll want to ask for automation so you can get regular reporting on what matters to you.

In addition to the above metrics, once this is in production, that might include:

- Volume of incoming data – how much processing it's doing
- Examples of low-confidence predictions, if your model outputs confidence scoı
- Examples of model errors



Recommended: llama3.1_newest_temp0.0

Bottom line: Find 8 out of 8 important items while only needing to review 26 documents instead of 100.

Key Metrics

Coverage 100%

Higher is better - finds more important content

Review Savings 74%

Higher is better - less content to review

Document Analysis Visualization

🟩 Found (8) 🟥 Missed (0) ▨ False positives (18)

This is part of an automated report visualization I built

For tracking model errors, you have an advantage when human review is part of yoı process: these interactions naturally generate labeled, ground-truth data that can bı sent back to your engineering team. However, if your system doesn't include autom labeling, deliberately add this step to your workflow with at least some of your new data. This ensures you can continuously validate your model and retrain when performance falls.

And by automating this monitoring, you get regular reports without creating additional work for your team.

Why To Have The Conversation

The hardest part of evaluating a model is making sure we're solving the right problem in the first place.

Your engineer wants to build something useful—but they need your help defining what "useful" actually means in your context.

If you find yourself nodding along to metrics without asking the questions that mat to you, that might make the conversation smoother in the short-term—but it won't either of you what you actually need.

So: ask what could go wrong. Spell out what success looks like. Make sure the evaluation you're getting answers the questions that keep *you* up at night.

3 Likes

Discussion about this post

Comments Restacks

Write a comment...