

i hate the future

ramblings about technology

Sunday, September 27, 2026

the normalization of inexplicable failures

In a recent episode of *President Curtis*, the President struggles with opening a door on two separate occasions.

These doors don't work because there are obstructions in the way: a body initially, then roughly a billion dollars worth of gold.



In both instances, in response to the frustration, the character mutters "stupid thing sucks." This is not a reasonable model of doors! Doors should not "suck" inexplicably! I found these moments outrageously hilarious¹ but maybe my stupid brain just sucks.

Jev: Making more doors that suck

The Internet has been abuzz about Jev, an AI model developed by TypeSafe AI, which returns typed values with probability estimates. The important things about Jev are, as far as I can tell:

- it is fast and cheap,
- you can build on it quickly,
- it is fast, and
- it is cheap.

I'm not particularly good at understanding what technology will get adopted.

I still don't understand² Slack.

But this product is even more confusing to me. This isn't "[getting an FTP account, mounting it locally with curlftpfs, and then using SVN or CVS on the mounted filesystem](#)" -- you still have to do the hard part. To know if Jev is working, you have to build evals and a ground-truth pipeline. If you have evals and a ground-truth pipeline, you're already most of the way to fine-tuning your own solution.

Wait.

Do you still have to do the hard part?

Maybe my problem is expecting products to work.

Nobody buying this is running evals. They're just handing opaque questions to Jev and getting opaque responses. Charitably, this allows them to check the "AI-powered" box and ship before Friday, and when this breaks downstream logic, they can always shrug and say "well, AI makes mistakes."

Error budgets? Failure modes? Test sets? All of those can be handled later. The user can discover the failure rate! You've already shipped!

False Confidence

"Oh," the strawman responding to my post responds, "you haven't considered the fact that Jev gives you *confidence scores*!"

What are you going to do with those?

For you to do something reasonable with confidence scores you need to have both an understanding of the calibration of those confidence scores and also a model for the costs of the uncertainty.

On the calibration side: Jev's topline ad copy is mostly about how well they score on various benchmarks, but not about how calibrated their confidence scores are. There's a [cookbook about using confidence scores to go up a tree of classification](#) but that's fundamentally not about how good the confidence scores are.

On the modeling side: nobody wants to think about this. It always ends up being "Ehh, let's just accept any response over a threshold. 0.9 sounds about right? What's for lunch?" Their [documentation itself makes up a 0.5 threshold for "do nothing" and a 0.9 threshold for "do high-risk actions"](#) [\[archive\]](#) with an aside info box stating "the correct threshold values depend on your domain and the performance of the model for your use case."

At best, people use confidence scores in a cargo cult manner. At worst, people use them as an excuse for why the API call failed. The model was only 73% confident! That means my error budget is 27%!

Accountability

When a button breaks on a website, I have a model about what should have happened. Somewhere a contract got broken. My DNS is broken. Somebody shipped some slop that has JavaScript syntax errors along only a certain path. A handler threw that wasn't expected to throw. I might not have access to debug just an HTTP status 500, but I expect there to be somebody whose job is to understand why the endpoint is 500ing. The ownership is well-defined albeit opaque³.

For many users, however, the actual experience is roughly just "stupid thing sucks." Software already feels capricious; more failures just change the rate of frustration. It seems like not much of a loss to remove the possibility of following a failure to a concrete cause. Sometimes things just suck.

This leads to a normalization of inexplicability.

My fear is not that more things will fail when things are accelerated by LLM-driven development. They will. They have. Such is part of the price of building things in a novel manner.

My fear is that "sometimes it just sucks" is going to be more and more the accepted endpoint of investigations. This is sad because LLM-accelerated development can indeed *help us* solve some of these issues. There are plenty of automated QA workflows that aren't written because of lack of engineering time. The very eval that would get you most of the way to replacing (or even justifying the use of) Jev can be a few prompts away.

The tragedy of software engineering today is that we are actively engineering systems where *neither* the user nor the builder seems to have any interest in checking whether or not there's a body behind the door.

We just shrug and conclude: *stupid thing sucks*.

¹This reminds me of a saying that I find similarly hilarious: "sometimes you get the elevator, sometimes you get the shaft." This is also not a reasonable model of elevators!!

²The lock-in network effect makes sense to me but I'm still bewildered as to how people standardized on a product that does not even reliably deliver messages. I have seen messages dropped on free, paid, and enterprise instances that only show up weeks later.

³Well, maybe "well-defined" is optimistic. After [Bill Gates famously failed to download Movie Maker](#), everybody agreed that it was presumably *somebody's* problem, just not necessarily theirs. Ideally we can get even this level of accountability without the customer being Bill Gates.

patrickxia at [8:24AM](#)

Share

No comments:

[Post a Comment](#)

[Home](#)

[>](#)

[View web version](#)

About Me

 [patrickxia](#)

[View my complete profile](#)

Powered by [Blogger](#).