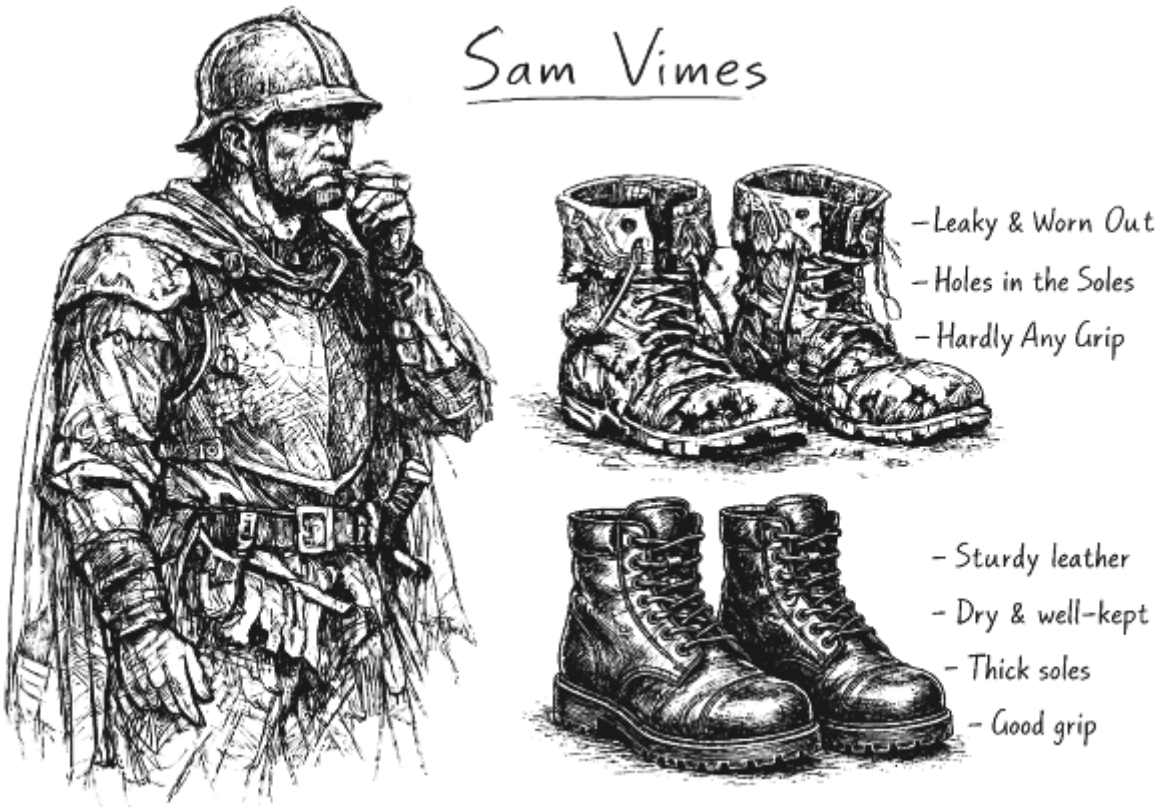
Vimes Boots & why the right AI evals could save your project

07 May 2026 · ai, automation, cloud, evals, illusion, investigation

“The reason the rich were so rich, Vimes reasoned, was because They managed to spend less money” - Sam Vimes (https://en.wikipedia.org/wiki/Sam_Vimes), from Men at Arms by Terry Pratchett.

The theory goes that richer people can afford a pair of boots that last longer before repair or replacement than poor people, who can only afford cheaper and less hardy boots requiring replacement in just a ⅓ of the time.

It’s the old “buy cheap buy twice” idea and it still applies today, especially in the world of AI agents and LLMs. Currently your agent is ‘managed’ or ‘directed’ at least in part by a harness. In fact, the IDE coding agent is really a harness + an LLM. The LLM “writes the code” and requests tool use, the harness enables those tool calls. E.g.: Write a file or run a command line tool etc.



In one sense the LLM does the productive side of the agents work while the harness has a more deterministic & feedback role. (e.g. reports back that the command the LLM suggested returned result xyz, or the code errored when we ran it)

That dynamic works well, and you can see that in Anthropic's toolset, where the LLM i.e.: Opus 4.7 and the harness i.e.: Claude Code has become hugely effective. Claude Code and other tools with agent capabilities fit together well. But these tools can deliver a slightly perverse incentive, depending on how you pay for your agents. If you pay a flat fee then this might be less of an issue, but if you pay per token, that might influence the vendor's motivations.

E.g.: What if your LLM isn't very good at what you are using it for? we've all seen the benchmarks, stating how well the LLMs are doing at some maths reasoning etc. But are they good at what you do? so if I think about banking, financial services or payments - does the LLM, create code that works well for your agents operating in your business environment?

Let's say we're using my new imaginary LLM: "VimesBootsLLM Pro" it's fast and it's 30% cheaper than the competition and it has "reasoning" and tool use etc. It also passes all the usual benchmarks and is up there with the best of them in the league tables.

But what if you have your agent doing specific tasks (in e.g. banking or insurance) as part of their workflow? E.g.: You might have agent(s) writing code relevant to that domain – Does it perform well? And what does well/good mean here?

If we take a trivial example from bank payments, for example IBAN (International Bank Account Number) validation. Can it create the code repeatedly? How often / what percentage of its code is correct? (and yes, what do we mean by correct…) If our agent "harness" is using this model and finds code errors that terminate execution 50% of the time – the harness is going to be sending that code & result back to the LLM, which is going to rewrite that code maybe doubling the number of tokens used.

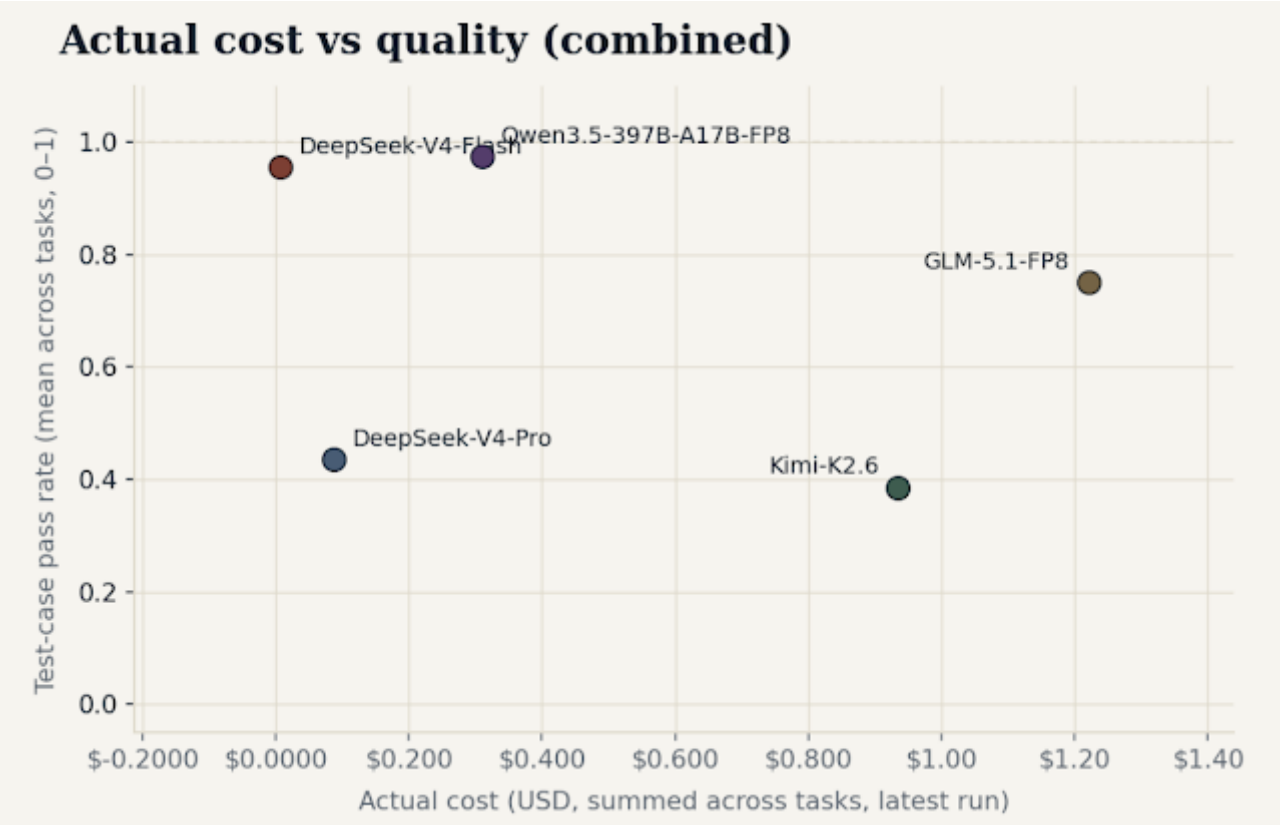
What if your tests/evals find the code does not error BUT also doesn't give the correct response 25% of the time? There are edge cases in even simple algorithms like IBAN validation. Then the harness will send this back to the server and the LLM will try again. All this back and forth is costing you money and cluttering your context window (The agent's short-term memory).

As you can see the headline cost and benchmark stats are like mileage or battery life stats YMMV - Your Mileage May Vary. What's even more misleading is that "reasoning" models can spend a lot of time just trying internally trying to get to the right answer. That internal reasoning uses tokens and while the price per token cost may be low, if your model is doing twice as much reasoning the cost will be proportionally more, than an LLM that does less "reasoning".

Thinking this through you can see that it's in a vendor's interest to maybe not be quite as good at one-shot coding (getting it right first time unaided by similar examples), if they are selling usage by the token. They maybe might create an awesome harness that can handle various values of fail, and handle it gracefully, and get the LLM to fix it next time around, especially if it's failing reasonably quietly and user is not inconvenienced. It's a bit like how it's alleged that some search engines reduced the quality of their search results in a misguided attempt to sell more adverts for a short term-spike in revenue & long-term total collapse of user trust. Or how dating apps are not incentivised to make especially good matches, because if they were almost perfect - you are less likely to keep using them (as you will be dating your one true love – that you met on the first match!)

I'm not suggesting this is Anthropic's plan, in fact I think they take measures to ensure this isn't the case for Claude Code users (their recent doubling of allowances is consistent with this viewpoint). But if we look at the industry as a whole and factor in not just IDE based coding agents, but also look at other AI agents, running headless in your back-end systems, are the model maker's incentives fully aligned with yours? Measuring the cost per successful delivery is a better bet.

Like Vimes and his cheap boots problem, sometimes we need to look at the total cost of ownership over time & how we intend to use the LLM. And it makes sense to look at this without custom skills and harnesses to reduce noise. I don't mean to say that such agent & skills checks should not be done – they should be done, it's just they are testing something else, something bigger, and are more akin to end-to-end tests.



DeepSeek V4 Flash does almost as well as Qwen 3.5 at a much lower cost.

Take this set of trivial “eval” results for some payment related functions. Essentially the LLM is provided with a typical bank payment validation functional description and then asked to create code to-implement it. Then the code is unit tested with several examples and the results recorded. Rinse and repeat many times across many function descriptions and LLMs. We then factor in the cost and accuracy, and we can then get a pretty-decent idea of how well the LLMs handle the bank payments domain.

This sort of analysis is going to be of greater concern in the coming months as more teams adopt agentic approaches to solving the problems and keep wondering why they have such high inference costs despite choosing what they thought was a good model and low price! Smart companies will look for this sort of analysis to ensure they don’t buy cheap and buy twice.

This sort of analysis of software, of AI tools and teams is what I do, if you want to know more - get in touch (https://www.linkedin.com/in/peter-houghton-374a36/).