The Chat Box Is a Language Model on Your Machine

A 1.2B model runs in your browser and answers questions from this site, with no server and no API key. How the retrieval works, what it costs, and which models did not make the cut.

july 26, 2026 · Projects

---

there is a chat box on this site. when you open it, your browser downloads a 760mb language model and runs it on your gpu. nothing is sent anywhere, because there is no server to send it to. it answers questions about me from the pages you are already reading.

this is how it works, and what the numbers look like.

the shape of it

the first version put my entire cv, about 4,500 tokens, in front of every question. it also ran two throwaway yes/no generations before each answer, to decide whether the question was on-topic at all. three model calls per question, and it knew nothing about the blog or the projects page.

now the site is chunked into 95 passages at build time and embedded into a 111kb file that ships with the page. your browser embeds only your question, one forward pass over about fifteen tokens and roughly two milliseconds, then searches that index.

the search is hybrid, and it needs to be. this corpus is one person's life, so everything in it is semantically adjacent to everything else; a 384-dimension vector cannot reliably separate "musiio" from "influize". exact term matching carries the proper nouns, embeddings carry the paraphrases ("where did he go to school"), and reciprocal rank fusion combines them without needing the two score scales to be comparable, which they are not.

then there is a gate. if nothing in the index is close enough to the question, there is nothing to ground an answer in, so it says so, in about a tenth of a second and without running the model. that gate replaced both classifier calls and is more accurate than they were. three generations per turn became one.

what reaches the model is a short instruction block, the passages that came back, and your question. it cites what it used, and those citations become links under the answer.

what it costs

download, once	760mb, cached after
cold load	~21s
search	~26ms
reading the prompt	~830ms
writing the answer	~540ms
a refused question	~0.1s

prefill dominates decode here, about 2:1. tokens per second is the number usually quoted for these models, and for a grounded chat it is the smaller cost.

the cache

the instruction block is identical on every turn, so it is run through the model once at load and its key-value cache is reused. that is worth about 420ms a turn, a quarter of the total.

the cached part was 975 tokens and stayed 975 tokens no matter how long the conversation ran, so the _share_ of each prompt it covered went down as you talked, from 55% to 51%. a cache that stops growing matters less the longer the conversation runs.

a cache can only skip a prefix that matches token for token, so what it can cover is decided by the order the prompt is assembled in. more of the prompt is fixed than it first appears: earlier questions have their retrieved passages stripped out, and earlier answers have their citation markers removed, and both of those edits happen exactly once, when a turn stops being the current one. after that the history is frozen, so it can all be carried forward, and the cached region grows by one exchange per turn instead of standing still. seven turns in it covers 1,290 tokens rather than 975, and the conversation is prefilling about a quarter fewer tokens than it was.

two other arrangements measure worse. keeping every turn's passages in the prompt caches the most, but spends the savings carrying stale context, which is what used to make it answer turn four out of turn one's passages. pinning the passages to a fixed position so a repeated search matches came out no better than doing nothing, because retrieval has to return the same set in the same order for that to pay, which is rare in practice.

model selection

smaller is not faster. a model with half the parameters ran two to four times slower, because it wrote several hundred words where the larger one writes forty. decoding cost is per token, so verbosity dominates parameter count.

the fast one fabricates. a 230m model loads in nine seconds instead of twenty-one and answers in under a second, but asked whether i know a language that appears nowhere in my skills list, with that list in its context, it says yes. asked whether i worked at a company i never worked at,

it says yes to that too. few-shot examples showing it declining exactly that kind of question, two hundred tokens above the point of use, did not change the behavior. at this size the model accepts the premise of whatever it is asked.

the same examples degraded the larger model. one of them mentioned musiio, and that was enough for musiio to start appearing in unrelated answers.

reasoning does not help. the step-by-step variant of the same model scored worse in every category and took four times as long, spending 658 tokens reasoning before the visitor saw a word. reading four retrieved passages is not a reasoning problem. the answer is already in the context.

so the lineup is one model.

the evals

there is a graded battery of 68 questions covering grounded lookups, multi-passage synthesis, follow-ups that depend on the previous turn, questions built on false premises, things that must be refused, and things that must _not_ be refused.

the first version of the battery had twelve cases and the model passed all of them, which made it useless for comparing versions. a saturated test can only show that something broke.

expanding it exposed a class of failure the small set never touched: false premises. asked where i got my mba (i don't have one), the model reported a doctorate i never finished. asked how old i am, it worked it out from my job dates and offered "early thirties." asked why i left a company, it invented a motive and hedged it with "probably."

it currently scores 55/68. the remaining failures are dates it gets wrong, gibberish it answers instead of refusing, and a couple of roleplay prompts that talk it out of its job. each case is scored on a scale rather than pass/fail, with a note on every lost point.

what it still gets wrong

ask it "archanan?", just the word, and it recites where that company sits in my timeline instead of telling you what it was. the answer is coming from the career summary that sits in every prompt, and suppressing it breaks the follow-up questions the summary is there to serve.

if your browser has no webgpu, it does not work at all, and it says so. it can't fall back to your cpu, and the reason is narrower than "too slow", though it is also too slow. every compressed version of this model stores its vocabulary in a format the cpu engine cannot read. the gpu engine can, which is why one works and the other does not. a faster cpu engine would not fix it either: the best one available runs a model this size at two to five words a second.

it also will not tell you anything that is not on this site, which is most things. that is by design. it can tell you where i worked and what i have built. it cannot tell you what i think about your architecture. for that, email me.

https://alexnodeland.com/timeline/260726_in-browser-chat/