

Introducing System One Models and Jev

TypeSafe AI's company and product launch announcement.

Diogo Almeida, founder, TypeSafe

At OpenAI, I helped build the methods that made language models useful at following instructions and talking with people. That work became part of the path to ChatGPT.

My research there left me with a question I couldn't shake: if these models are so capable, why have they automated so little of the world's work? The usual answer is that the models need to get smarter but I think a large part of the answer is simpler than that: we gave AI the wrong interface.

Large language models produce strings, which are perfect when the output is meant to be read by a person. Yet strings don't make sense when the output is meant for software. Software is built with known types, explicit values, predictable latency, and auditable behavior.

After two years in stealth, countless technical challenges, and research breakthroughs... I am beyond excited to announce that today, TypeSafe AI is releasing our first **System One Model**: a new class of frontier models built to make fast, structured decisions that software can use directly.

Our first public model is **Jev**, available today in early access at the cost of \$42 per **billion** input tokens and \$0 for output tokens (yes, they're free).

Jev does not write essays, hold conversations, or generate code. Instead, Jev outputs decisions that can be directly ingested in code. Our model is 100x faster and 100x cheaper than existing LLMs, while achieving similar levels of intelligence (see evals section below).

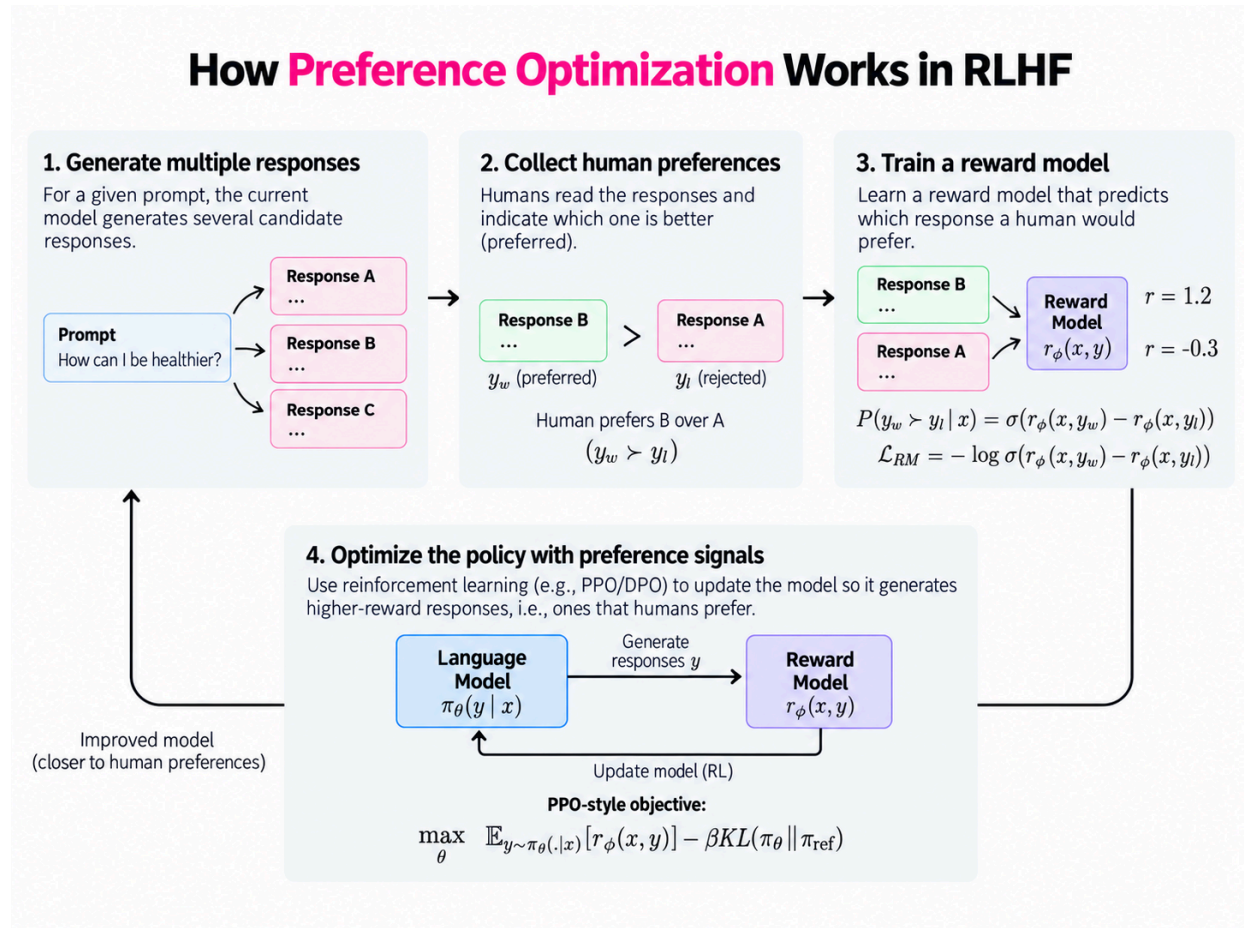
After two years in stealth, we're excited to finally introduce a new intelligence primitive for software.

From RLHF to RLCD: What the task changes

First, a quick history on the past ~6 years of AI research and how we got to the incredibly smart models we have today.

I co-invented RLHF (Reinforcement Learning from Human Feedback) and InstructGPT, the methods that lead to ChatGPT and GPT4. RLHF remains foundational to most modern LLMs: its preference-optimization methods were a breakthrough because they turned pre-trained LLMs from internet auto-complete machines into chatbot assistants.

But I am increasingly troubled by what it has taught the models to become. By training the models to please people, we gave them an incentive to tell us what we want to hear, even if wrong.



In plain English: RLHF gives rewards for outputs that people rate highly, without letting the model move too far from the original model.

The Bitterest Lesson: Doing the right task matters more than scale

In 2020, GPT-3 was the best language model in the world. It autocompleted your text, but it wouldn't answer your questions. So we took a model roughly 100x smaller, at GPT-2 scale, and gave it a different task: follow the instructions a human just typed. Then we optimized against human rankings of the outputs.

The result was a breakthrough. People preferred a tiny 1.3B parameter InstructGPT's answers to a 175B GPT-3's answers! Closing that gap through scale alone would have required something like GPT-15.

Most people's takeaway was that "RLHF works." But the more important lesson is that the choice of task was worth more than 100x the compute to train. This is what I call [the bitterest lesson](#): optimizing for the right task matters more than data, compute, or algorithm.

Every lab optimizes for the same task during RLHF: *produce the text that a human rater prefers*. That was the right task for a chat product, but it is the wrong task for automation.¹

¹ **Note on RLVR:** After OpenAI O1 was released in 2024, major labs have also been doing RLVR, which optimizes for correctness with a verifiable reward. While useful, I believe optimizing for calibrated decisions is the right task for automation.

Previous version:

- [Launch announcement blog post](#)

Training language models to follow instructions with human feedback

Long Ouyang* Jeff Wu* Xu Jiang* Diogo Almeida* Carroll L. Wainwright*

Pamela Mishkin* Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray

John Schulman Jacob Hilton Fraser Kelton Luke Miller Maddie Simens

Amanda Askell[†]

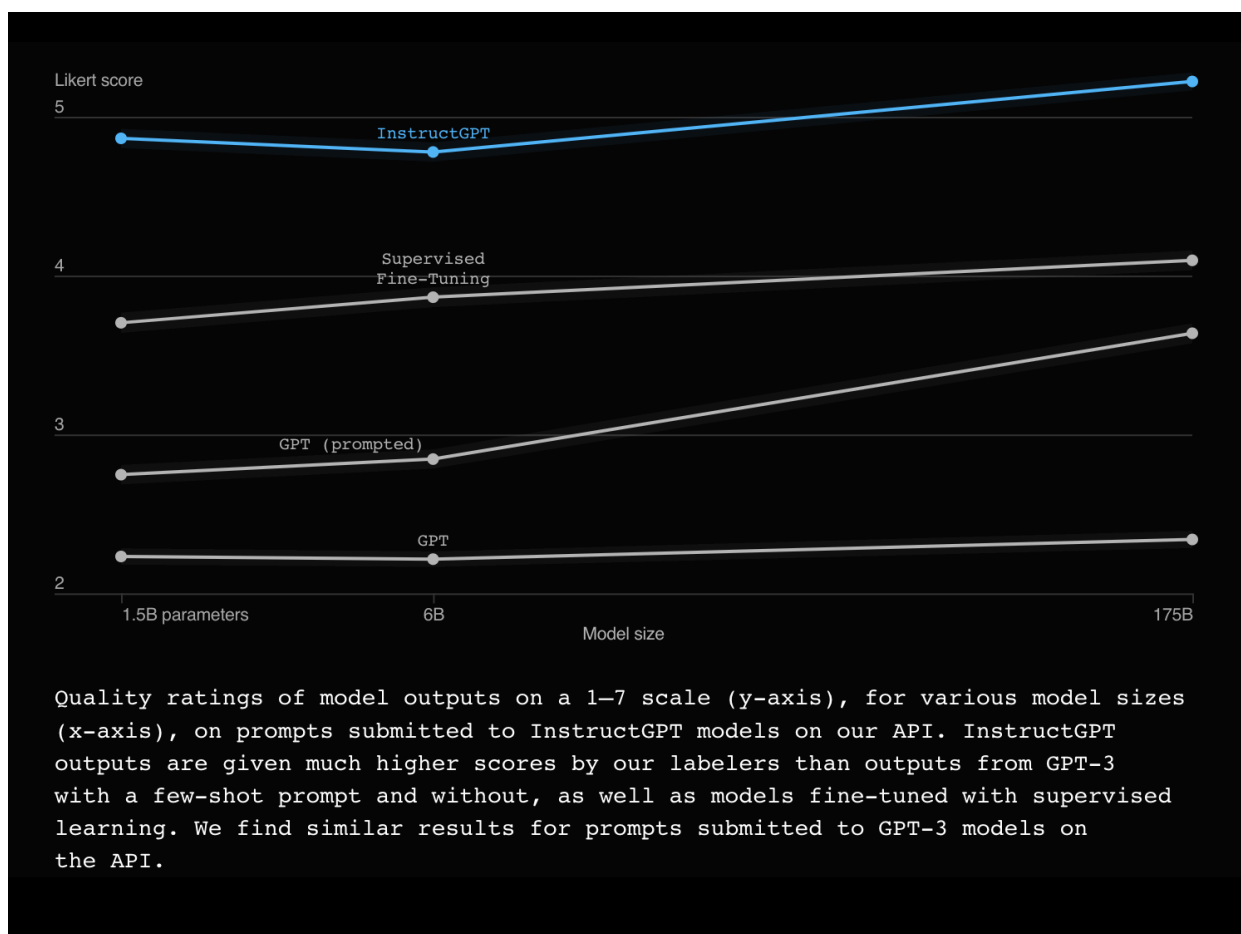
Peter Welinder

Paul Christiano^{*†}

Jan Leike*

Ryan Lowe*

OpenAI



A 1.3B parameter (top) InstructGPT model crushes a 175B pre-trained model at the task it was trained on: *human preference*. TypeSafe’s task is completely different: *calibrated decisions for automation*.

The Right Task for Automation: Calibrated Decisions

At TypeSafe, we spent the last two years building intelligence optimized for decisions and automation.

We started with the task: read some information, answer a specific question, and return a value software can use, along with probabilities over the possible answers. The judgments we care about are the ones a knowledgeable person can make quickly: what a customer is asking for, whether a document supports a claim, or how closely an item matches a description.

That choice of task led us to a new model architecture, a new decision sampler for selecting answers, and a training method we call **Reinforcement Learning for Calibrated Decisions (RLCD)**.

We call the resulting class of models **System One Models**.⁶

Optimizing for different tasks comes with different tradeoffs:

	LLMs / RLHF	System One / RLCD / TypeSafe
Optimizes for	<i>Human preference</i> : outputs that human raters prefer.	<i>Calibrated decisions</i> : answers with probabilities you can trust.
Sampling	Autoregressive . Generates one token at a time, each depending on the preceding tokens.	Parallel . Evaluates each question against the same input independently and returns answers.
Calibration	Prone to overconfidence . RLHF can reward confidence even when an answer is wrong or uncertain. RLVR optimizes for pure correctness.	Trained for calibration . Probabilities are optimized to reflect the underlying uncertainty of the problem.
Inputs	Text instructions . Reads text inputs in natural language. Data, state of the program, and instructions co-exist in a single prompt and fully unstructured.	Structured Json or Text . Structured state of your program, or the data you want to process. Instructions are structured questions about the state .
Outputs	Generated text . Produces answers in text. In order to use it with software, responses are usually parsed and validated. Text outputs are very flexible, but can include hallucination and there is always some risk that the AI goes off the rails.	Type-safe structured values . Possible output type and structure is defined in advance. The model only returns values allowed within that type. Answers are accompanied with calibrated probabilities and confidence scores. General but constrained to what's useful for software.
Consistency	Less consistent for identical inputs . Ask the exact same question twice, and autoregressive sampling can produce two different answers.	More consistent for identical inputs . The same question on the same model returns the same answer consistently.
Speed	End-to-end response time is 3 to 329 seconds for frontier models. ⁴ Fast enough for	End-to-end response time is 70ms-500ms for TypeSafe. This can range from 40x-1000x faster

	LLMs / RLHF	System One / RLCD / TypeSafe
	interfacing with humans, but a big bottleneck when integrating with code.	for the same levels of intelligence (but only for System One shaped queries).
Primary Use Cases	Assistance and Copilots. Chat, writing, and coding through one human-centric text interface. Very general and powerful, but generally requires co-piloting because their freedom also means they might go off the rails. Reasoning and Planning. System Two thinking: reasoning models can work through multi-step problems, plan and strategize. Writes code just-in-time to take actions and modify files. Coding agents Markdown based instruction following coding agents which can massively accelerate a software developer. Almost always requires human oversight.	Automation Software. Fast, System One decisions and judgments which can be used by software to make AI-systems cheap, fast, reliable, and observable enough to run a million times without human oversight. Does not generate text or code. Map-Reducing over Big Data. Turn petabytes of data into features and insights. Real-time Applications. 100ms speeds means you can use AI in your real-time applications. Verify Everything. Score, judge, verify, guardrail, and detect jailbreaks of LLM prompts, reasoning traces, and/or outputs.

The [evaluation below](#) compares intelligence and latency on the same decision workloads.

Use software to control AI, not the other way around

Modern agents execute **while** loops on markdown text instructions, writing software just-in-time to take actions. They're powerful and also might go off the rails at any moment (which is why they need sandboxes and humans to watch over them). For automation, I believe that your code needs to decide what happens next: which checks to run, when to act, and when to loop in a person.

Our model gives common-sense intelligence with probabilities designed for code to use as thresholds (like having smart if-statements). RLCD trains the model's probabilities against outcomes so the probabilities are actually calibrated. When a model assigns a probability of 0.9,

that outcome should occur roughly 90% of the time for those cases. Calibration is measured across groups of predictions; it does not guarantee that an individual answer is correct.

The GPT-4 Technical Report illustrates why the training objective matters. On the multiple-choice questions evaluated in its calibration plot, the pre-trained model’s confidence closely matched its accuracy. After RLHF, that calibration was worse [\[1\]](#).

GPT-4 Technical Report

OpenAI*

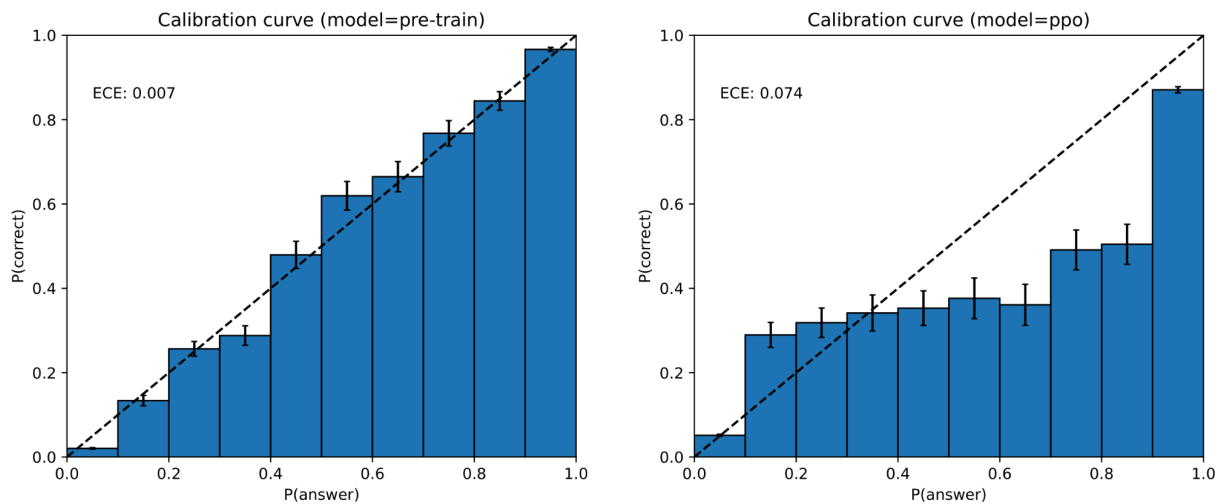


Figure 8. Left: Calibration plot of the pre-trained GPT-4 model on a subset of the MMLU dataset. On the x-axis are bins according to the model’s confidence (logprob) in each of the A/B/C/D choices for each question; on the y-axis is the accuracy within each bin. The dotted diagonal line represents perfect calibration. Right: Calibration plot of the post-trained GPT-4 model on the same subset of MMLU. The post-training hurts calibration significantly.

Consider a support request asking for a refund. You define the departments it can be routed to and ask whether the customer wants a refund, whether information is missing, and whether the request looks like a scam. Jev evaluates these questions independently in one request. The answers can be represented as:

```
None
{
  "route to": {
    "Refund Department": 0.95,
```

```
    "Technical Support": 0.03,  
    "Sales": 0.02  
  },  
  "probability requesting refund": 0.97,  
  "probability need more information": 0.10,  
  "probability scam": 0.04  
}
```

Your code leverages Jev's judgments to decide whether to route the request, ask for more information, issue an eligible refund, or send the case to a person. Meanwhile, deterministic, reliable code – not an agent – checks the order records, applies the refund policy, and calculates and issues the refund amount. You can require stronger evidence for issuing money than for routing a ticket!

Jev's probabilities and confidence allow you to measure performance on your data and choose individual thresholds for each action, based on the error rate your business can accept. Those thresholds determine how much work the system can handle automatically (no human in the loop) and which cases need review.

The first public System One Model: Jev

Jev is our first public model built around this interface. It is available today in early access, with input tokens priced at \$42 per billion and **free** output tokens.

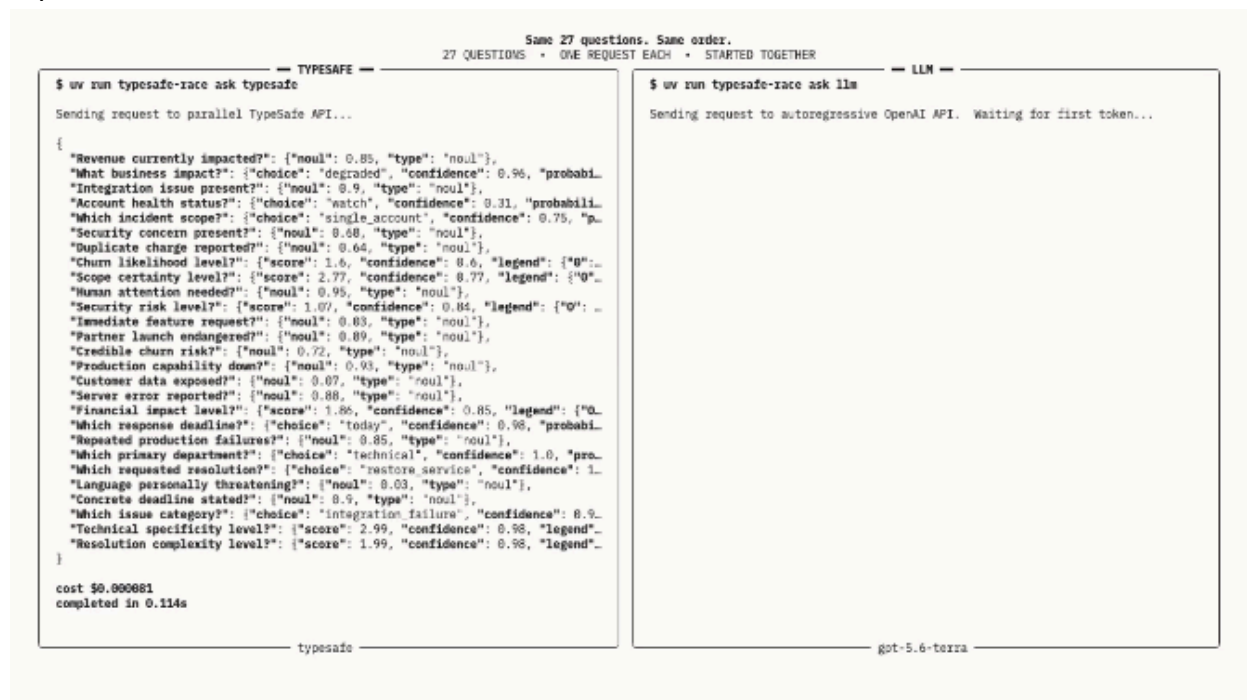
We named Jev after William Stanley Jevons. In the nineteenth century, Jevons observed that improvements in steam-engine efficiency increased the demand for coal. Greater efficiency made steam power useful in more places.

We expect a similar effect for machine intelligence. Every order of magnitude drop in the cost of intelligence unlocks orders of magnitude more use cases: checking every incoming message, matching every item in a catalog, or classifying every cell in a database.

At the same time, we're removing the time cost of intelligence. A person might happily wait several seconds for a thoughtful chatbot answer. Software making thousands of small decisions every second can't afford to spend several seconds at each step waiting for a written response.

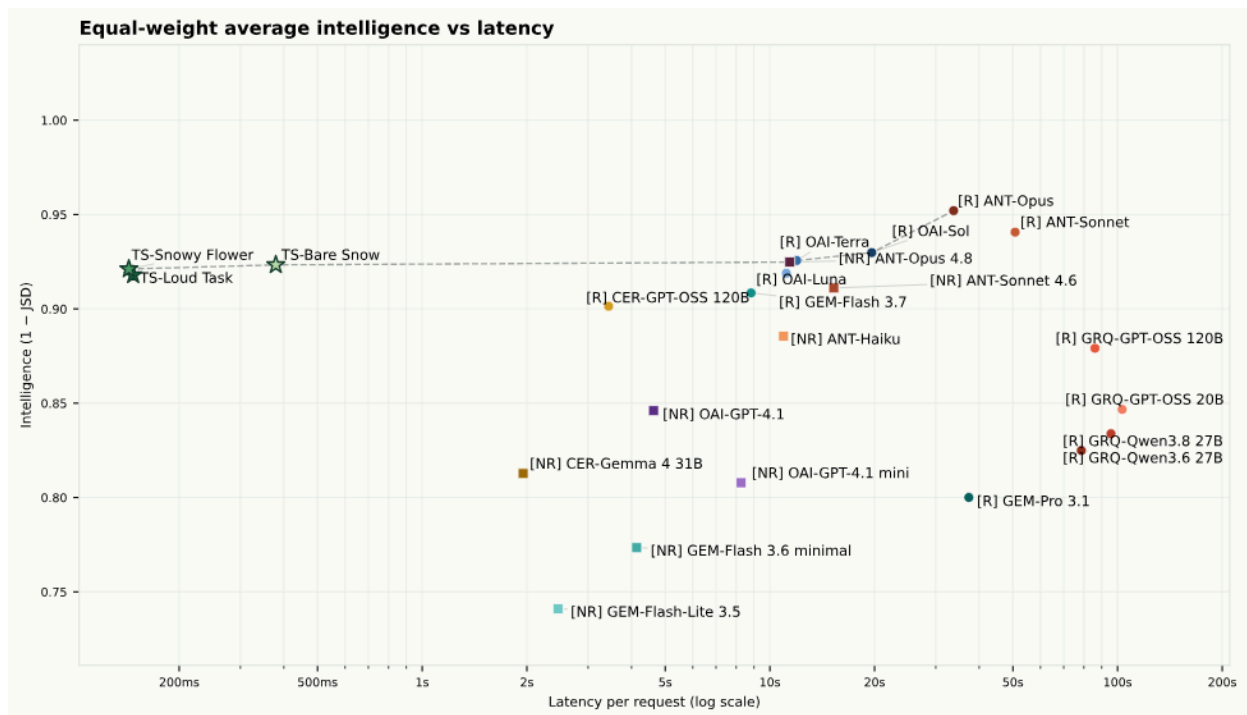
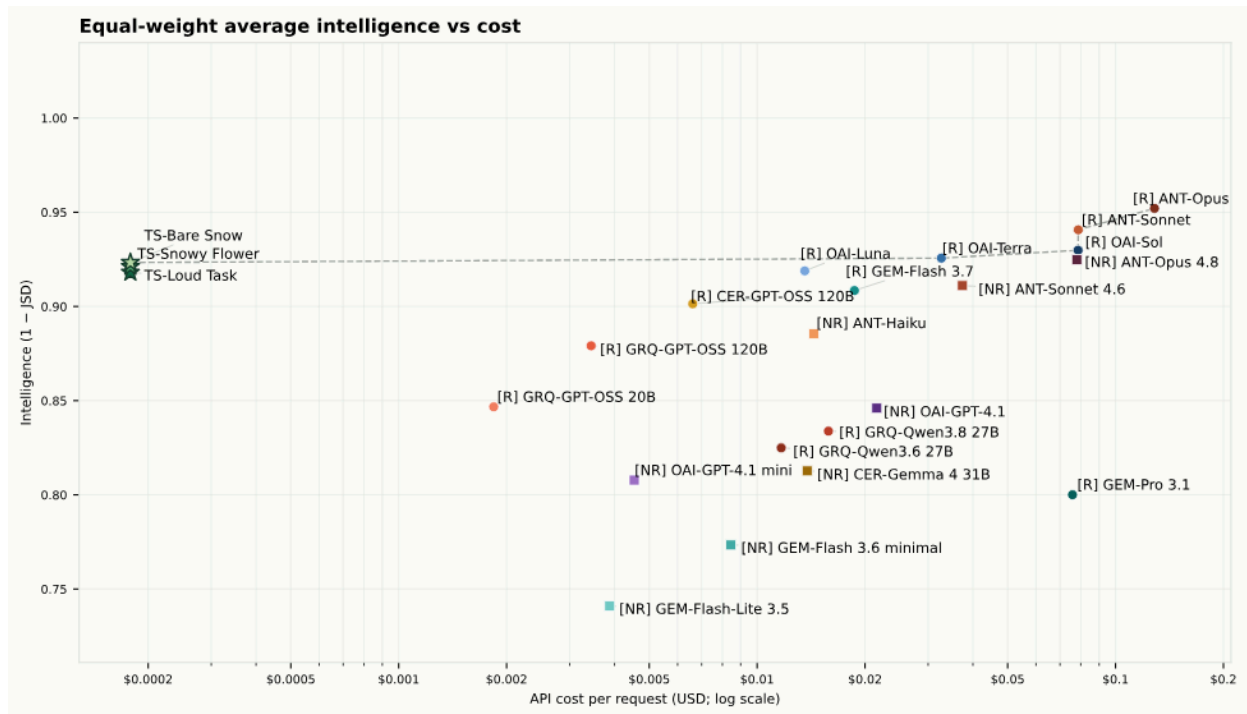
Our bet is that much of the demand for AI is unrealized due to these costs. Jev is an early step toward making intelligence an ordinary part of software.

<https://www.loom.com/1b92d161-35ae-4e95-b3a2-ff5fd5b48d41>



Evals: intelligence, cost, and speed

We care about the quality of a decision and what it costs to make it. The comparison below evaluates models on 10 documents drawn from a dataset of workplace tasks, with 15 decision questions per document.



Each labeled point is an evaluated model. Stars mark TypeSafe models; [R] and [NR] identify reasoning and non-reasoning LLMs. Higher and farther left is better. Cost and latency are per request containing 15 questions, and both horizontal axes use logarithmic scales.

On this evaluation, the TypeSafe models scored about 0.92 on the plotted measure, took about 0.14–0.38 seconds per request, and cost about \$0.00018 per request. Several LLMs reached similar scores, while the highest-scoring models did better at greater cost and latency.

The chart calls its measure “intelligence.” It measures how closely a model’s probabilities across all possible answers match a reference assembled from other providers’ models. We calculate similarity as one minus Jensen–Shannon divergence, a measure of how much two probability distributions differ. The evaluation gives equal weight to each category of decision tasks.

For each provider, we build the reference by averaging predictions from the strongest evaluated model at every other provider. That excludes a provider from its own reference, but also means different providers are measured against different references. A score of 0.92 is a similarity score, not 92% accuracy against human labels.

This is an early comparison on a small sample. It shows the cost and speed tradeoff on these decision tasks; it does not establish performance on every workflow or measure calibration. Developers should evaluate the decisions and error rates that matter in their own applications.

What’s next

Jev is the first System One Model, and we are still early. We have spent two years developing the architecture, sampler, and training algorithm, and there is a lot left to improve.

Today, we are opening early access and bringing developers off the waitlist as quickly as we can. We want to hear which decisions you need to automate, where Jev works, and where it falls short.

[Request early access to Jev](#) and tell us what you want to build. We started TypeSafe because we believe that AI needs an interface software could depend on. We’re excited to see what you build with Jev.

FAQ

TO BE ADDED

References

1. OpenAI, [GPT-4 Technical Report](#).
2. OpenAI, [Aligning language models to follow instructions](#).
3. OpenAI, [Training language models to follow instructions with human feedback](#).
4. [LLM benchmark explorer](#).
5. Daniel Kahneman, [Thinking, Fast and Slow](#). Our name draws on the distinction between fast, intuitive System 1 thinking and slow, deliberate System 2 reasoning.