SEP 24, 2026

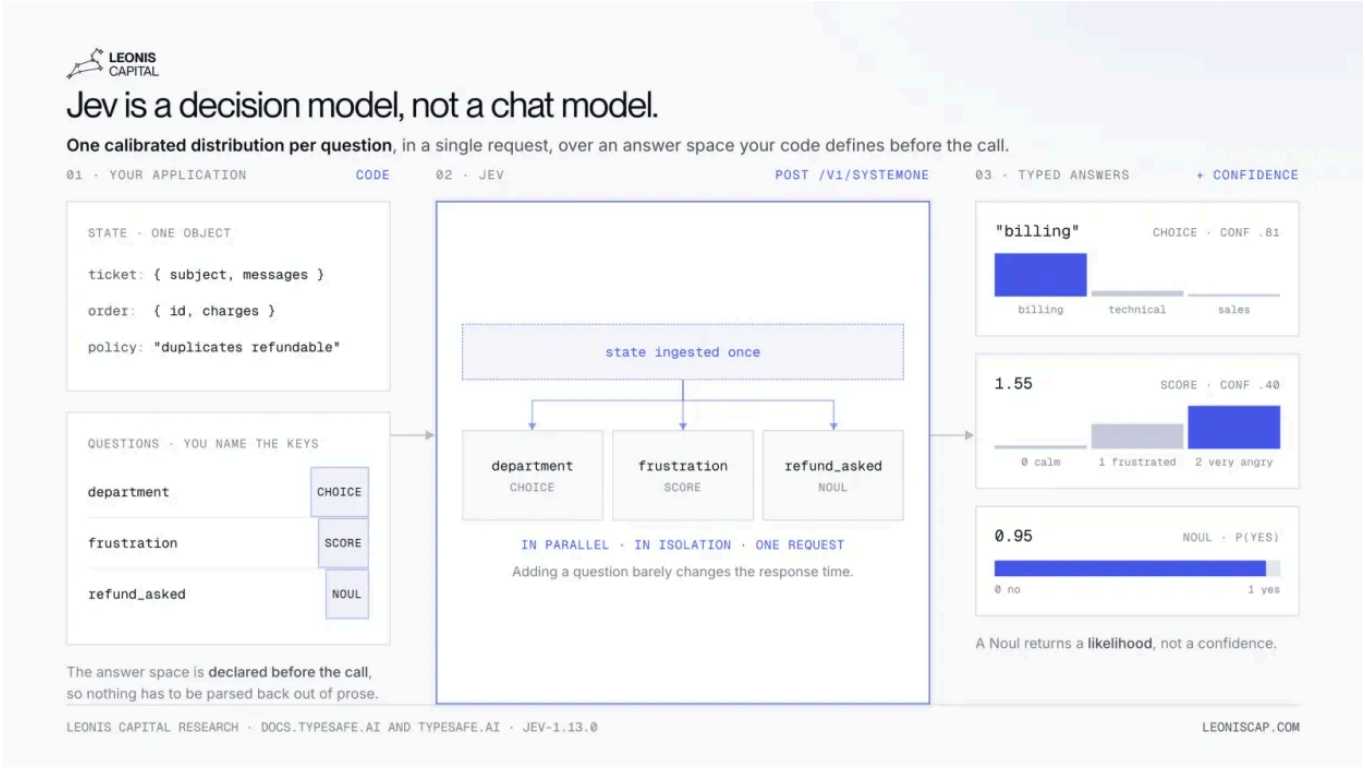
Jev and the Rise of Decision Models

Building for AI as a user, and not just humans, unlocks the next generation of AI applications



By Jenny Xiao, Liang Wu, and Jay Zhao

Leonis Capital · leoniscap.com/research/jev-and-the-rise-of-decision-models/

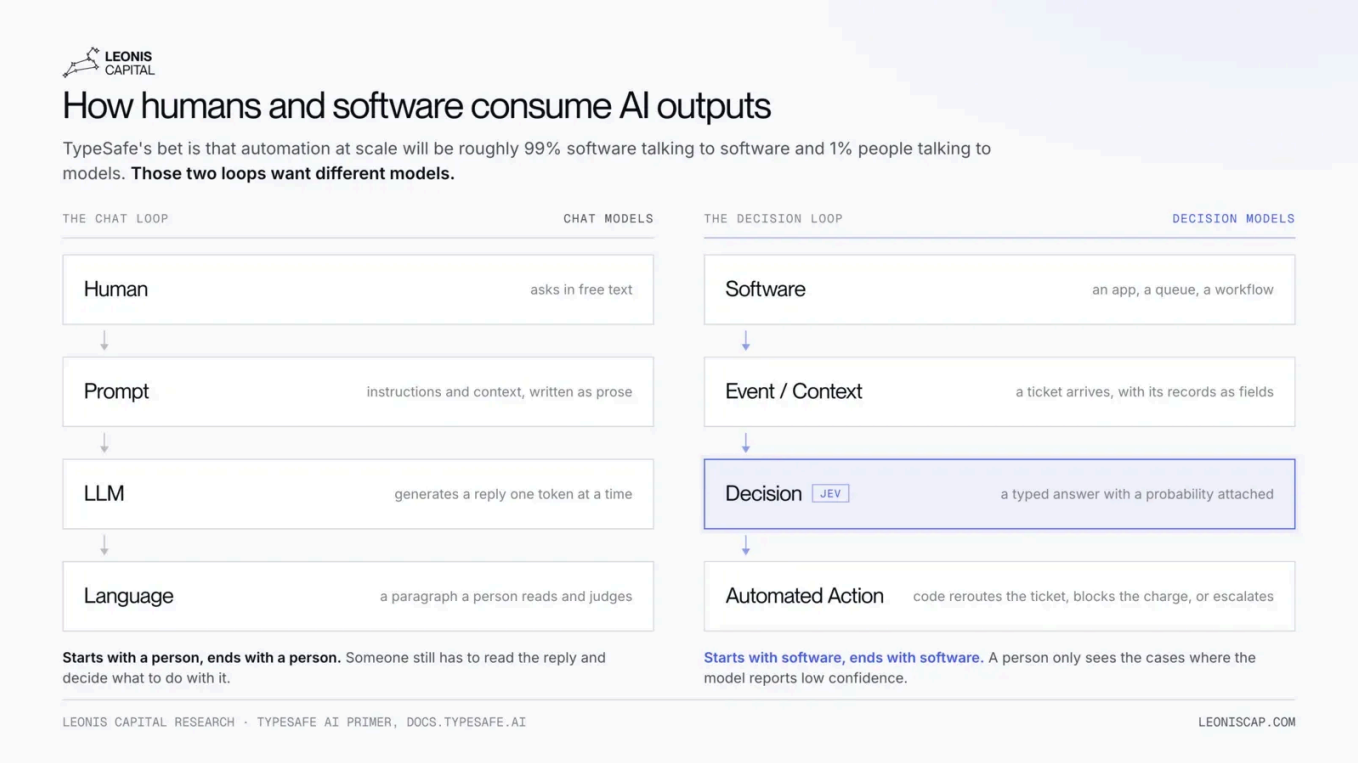


For the past three years, AI progress has mostly meant making general-purpose models better with smarter reasoning, longer context, better coding, and more natural conversation. That trajectory made sense because **the primary user was human**. We give models language, and they give us language back.

But increasingly, humans are not the only users of AI. Software is becoming a user too.

Unlike humans, software does not need an explanation, a reasoning trace, or an essay. It needs to know which queue a ticket belongs in, whether a transaction should be flagged, which tool an

agent should call, or whether an action should proceed. **Humans consume language, software consumes decisions.**



Jev is built around that distinction. Introduced by TypeSafe AI on September 15, the decision model became the fastest-adopted launch in Vercel's AI Gateway history, reaching more than twice as many paid teams in its first 24 hours as any previous model launch. What makes Jev unusual is precisely what it does not do. **It doesn't chat but returns structured decisions.**

That raises a more fundamental question about how we build AI for software: Should we carry the conversational interface of LLMs into automation? Why generate language at all if the application only needs the decision underneath it?

General-purpose LLMs became the default because they were versatile enough to absorb many different jobs. Jev suggests some of those jobs (classification, routing, scoring, and verification) may be better served by decision models optimized specifically for software.

To understand why, we first need to understand what Jev actually does differently.

Jev is Not Another LLM

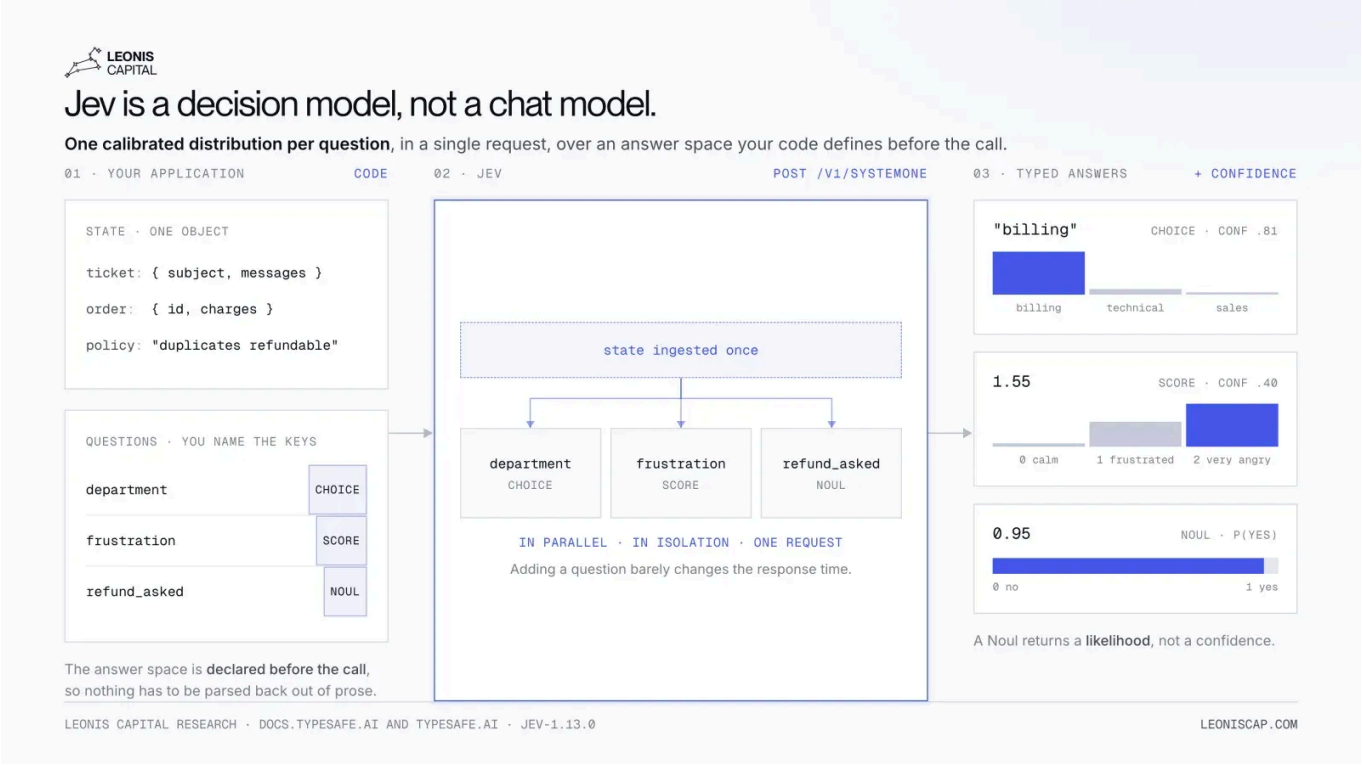
Some people might confuse Jev with simply another LLM optimized for a different benchmark, it isn't.

Most frontier models are built around a common flow and objective: given some context, predict the next token. Whether the task is writing an essay, generating code, or classifying an email, the model ultimately communicates its answer by producing language.

Imagine asking an LLM to determine whether an email is legitimate, spam, or phishing.

```
Email
    ↓
LLM
    ↓
"This appears to be a phishing email because it uses urgency
to solicit your password..."
```

Even if the application only needs the answer “phishing,” an LLM still generates an explanation token by token before software extracts the final decision.



Jev approaches the problem differently. Instead of asking the model to generate an answer, the developer defines the possible answers ahead of time. Jev evaluates the predefined choices directly and returns a probability distribution rather than writing a response:

```JSON
{
  "category": {
    "legitimate": 0.01,
    "spam": 0.02,
    "phishing": 0.97
  },
  "selected": "phishing"
}
```

Software can immediately consume this output. There is no explanation to parse, no free-form text to interpret, and no requirement that the model express its reasoning in natural language. **The distinction may appear subtle,** but it fundamentally changes what the model is optimized to do.

From Slow Generation to Fast Decisions

Autoregressive language models (i.e., most LLMs) generate output sequentially. If a model responds “This ticket appears to be high priority,” it must generate one token at a time.

```
Input:
"Classify this customer support ticket."

LLM output (autoregressive decoding):

Token 1  → "This"
Token 2  → "ticket"
Token 3  → "appears"
Token 4  → "to"
Token 5  → "be"
Token 6  → "high"
Token 7  → "priority"
Token 8  → "."
```

Each new token depends on the tokens generated before it. That sequential decoding is what makes LLMs extraordinarily flexible, but it also adds **latency to tasks where the application never needed prose in the first place**. Jev removes that long generation step. Rather than composing a sentence, Jev evaluates the available decision space. Multiple independent questions about the same context can also be evaluated in parallel. The speed is much closer to an API call than a chatbot response.

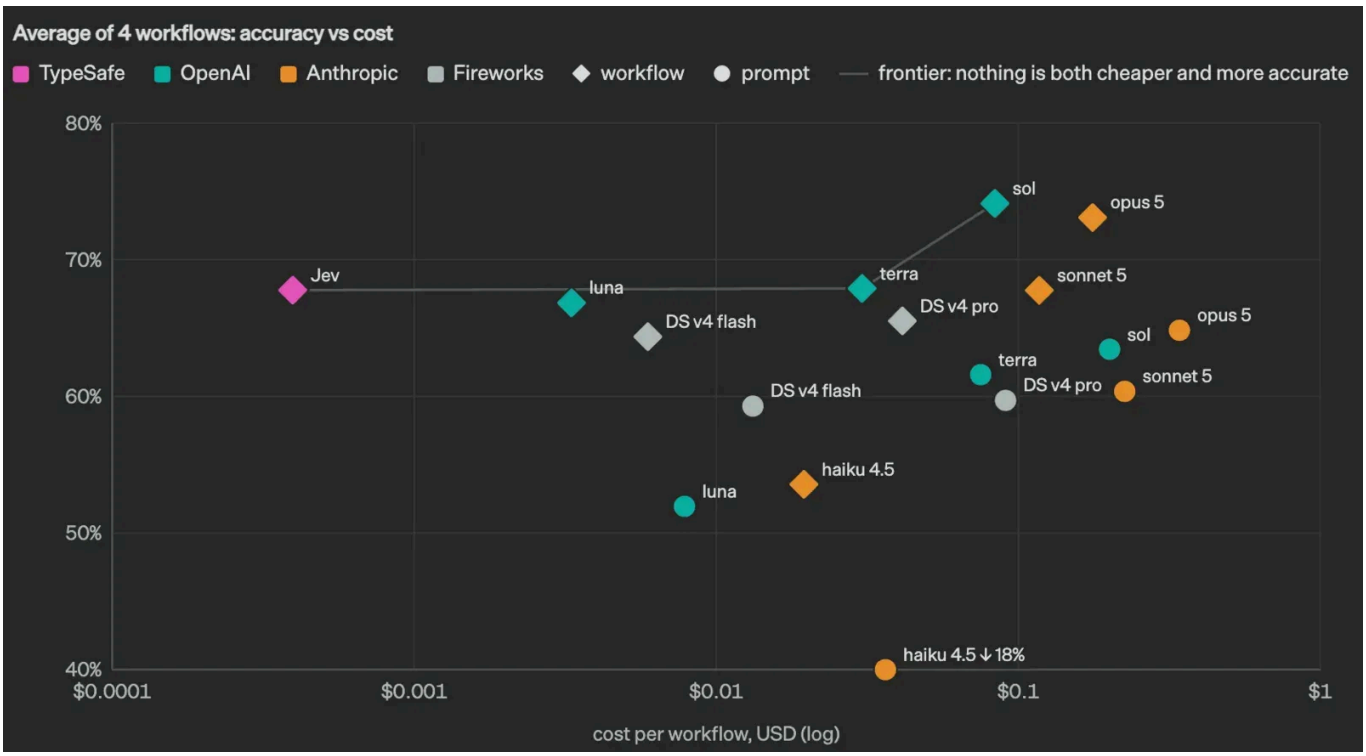
In a lot of use cases, **latency can matter as much as raw model capability.** A general-purpose LLM's response time depends on the model, the amount of reasoning it performs, and how much output it has to generate. Jev has no long autoregressive decoding stage. TypeSafe reports end-to-end response times of roughly **70–500 milliseconds**, fast enough for decisions to sit inside interactive software loops rather than only behind deliberate user prompts.

To illustrate this point, we built a demo where Jev is playing a real-time strategy (RTS) game *Age of Empires*. We had previously tried the same idea with LLMs, including low-reasoning-effort models, **but even the fastest models typically take around one to two seconds per decision, which is slow when the use case requires anything "real-time"**. By the time the model decided what to do, the game state had already changed. Jev makes this kind of loop practical because decisions return fast enough to keep pace with the environment.

Making Decisions Economical at Scale

Speed is only half of the equation. The same architecture also changes the economics of inference.

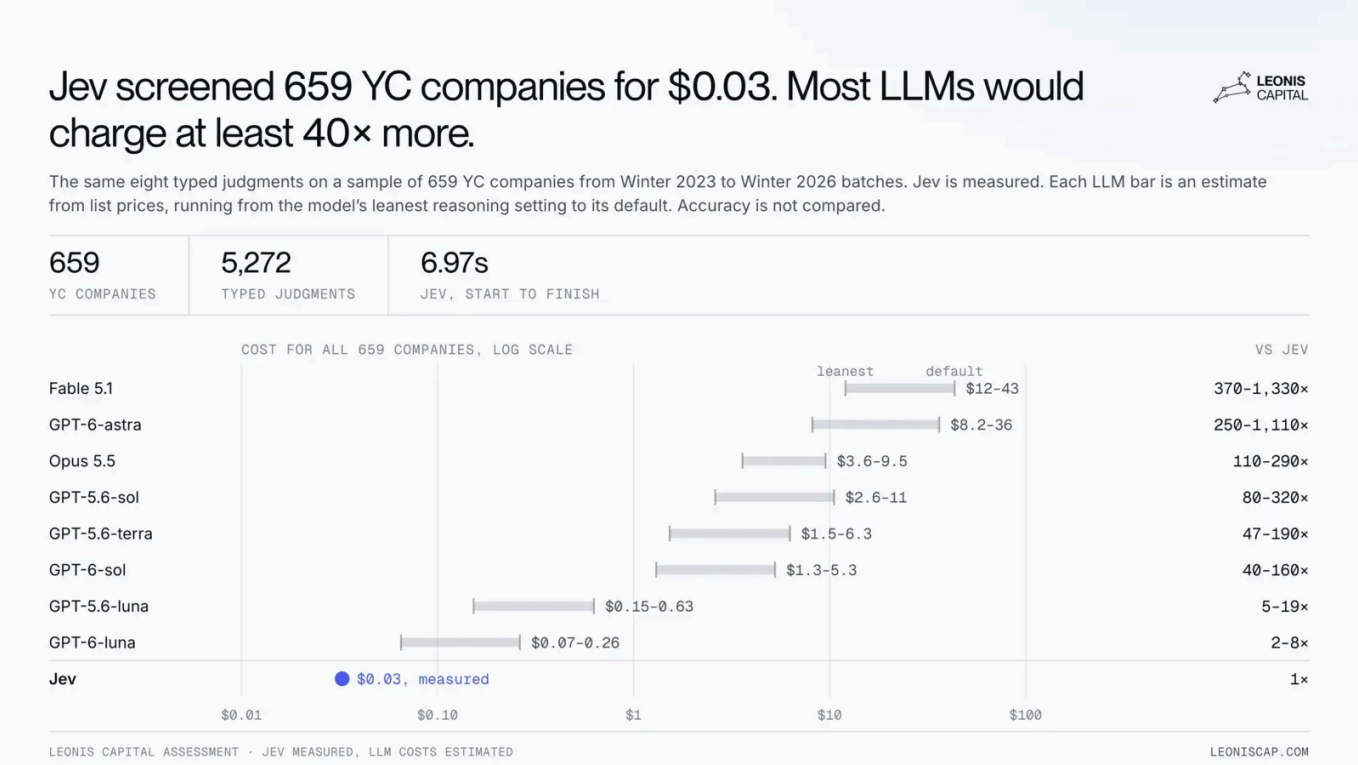
General-purpose LLMs charge for processing the input and generating the output. For a single classification, that cost may be negligible. But many useful applications require applying the same judgment hundreds, thousands, or millions of times. At that scale, inference cost determines whether the workflow is practical at all.



According to TypeSafe's benchmarks, Jev achieves roughly the same accuracy as GPT-5.6 Terra and Luna **while operating between 1/10 and 1/100 of their cost**. This puts particular pressure on smaller, cheaper general-purpose models whose role has been to handle the routine work that does not justify a frontier-model call. If specialized decision models can perform those same jobs faster and more cheaply, **some of that workload may move away from general-purpose models altogether**.

We tested this by building a screener for a sample of 659 YC companies from the 2023 through 2026 batches. For each company, Jev evaluated eight investment criteria that we fed it (including technical depth and market clarity) and returned structured scores we could compare across the cohort.

It screened all 659 companies in seven seconds for $0.03. Most LLMs would have cost at least 40x more for the same task. And the answers that Jev generated also held up against a human spot check. For example, fusion reactors for ships and hydrogen-electric aircraft ranked highest on technical depth.



Training for Calibrated Decisions

Jev is also trained to achieve a different objective.

The dominant post-training paradigms for frontier models have been optimized for human preference or verifiable correctness. **RLHF** rewards responses that humans prefer, which is well suited to conversational assistants. **RLVR** rewards answers whose correctness can be verified, making it particularly useful for domains such as mathematics and code.

TypeSafe calls its approach **Reinforcement Learning for Calibrated Decisions (RLCD).** The objective is not just to choose the correct answer, but to make the probabilities attached to those answers meaningful.

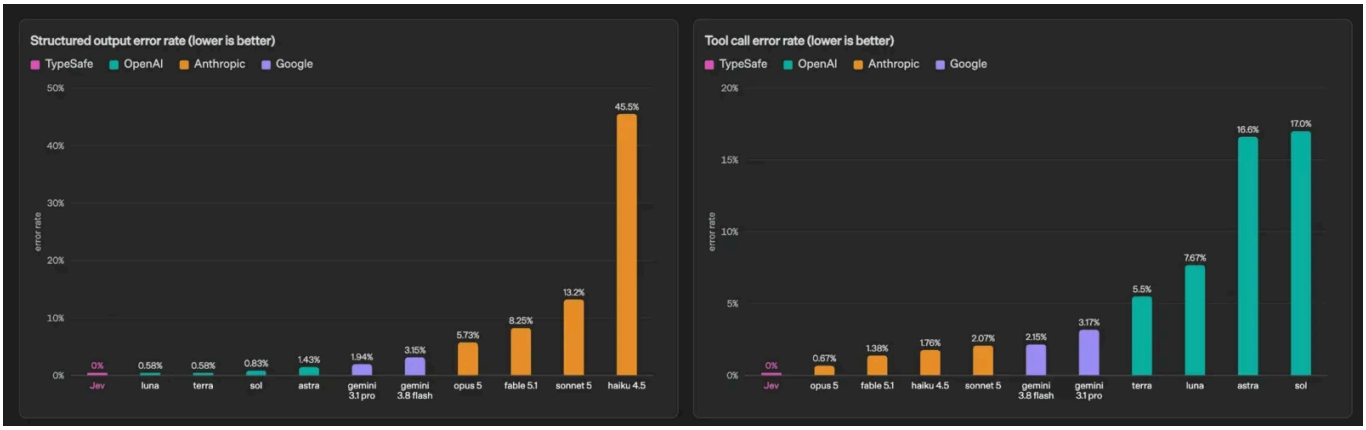
If a calibrated model assigns 80% probability to an event across many comparable cases, that event should occur roughly 80% of the time. Calibration does not mean the model is always right. It means that its uncertainty should correspond to empirical reality.

That matters because software can use uncertainty as part of its own logic:

```
> 95% confidence  → automate
70–95%            → send to a stronger model
< 70%             → escalate to a human
```

A Smaller Output Space

Jev reduces structural failures by restricting answers to a predefined schema. If the allowed categories are “legitimate,” “spam,” and “phishing,” an invented category such as “suspicious-ish” is not a possible output. Modern structured-output APIs have made LLMs far more reliable, but errors still happen, especially with smaller, cheaper models, and those mistakes can compound across multi-step workflows. In TypeSafe’s published comparison, Jev reports a 0% type-error rate, reflecting the advantage of constraining the output space from the start.



That does not mean Jev cannot be wrong. It can still classify a phishing email as legitimate, even with high confidence. Its advantage is structural, but it’s not infallible. This also means that Jev is only as good as the decision space users define. If the developer chooses the wrong categories, omits an important option, or writes an ambiguous question, the model cannot escape the schema and invent the missing answer. The same constraint that improves reliability also puts more responsibility on the application designer.

At the same time, Jev is **weaker on tasks requiring explicit computation or multi-step inference.** Public documentation and third-party testing flag counting, arithmetic, date comparisons, indirect/multi-hop questions, and noisy context as weak spots. A better architecture is often to leave deterministic computation to code and use Jev for the probabilistic judgment around it.

That limitation matters most in safety-critical settings. Defense and military systems are an obvious conceptual fit for fast, bounded decisions under uncertainty, but the required accuracy and robustness bar is far higher. Jev’s current public evidence does not yet demonstrate that it is suitable for autonomous, high-consequence decisions in those environments.

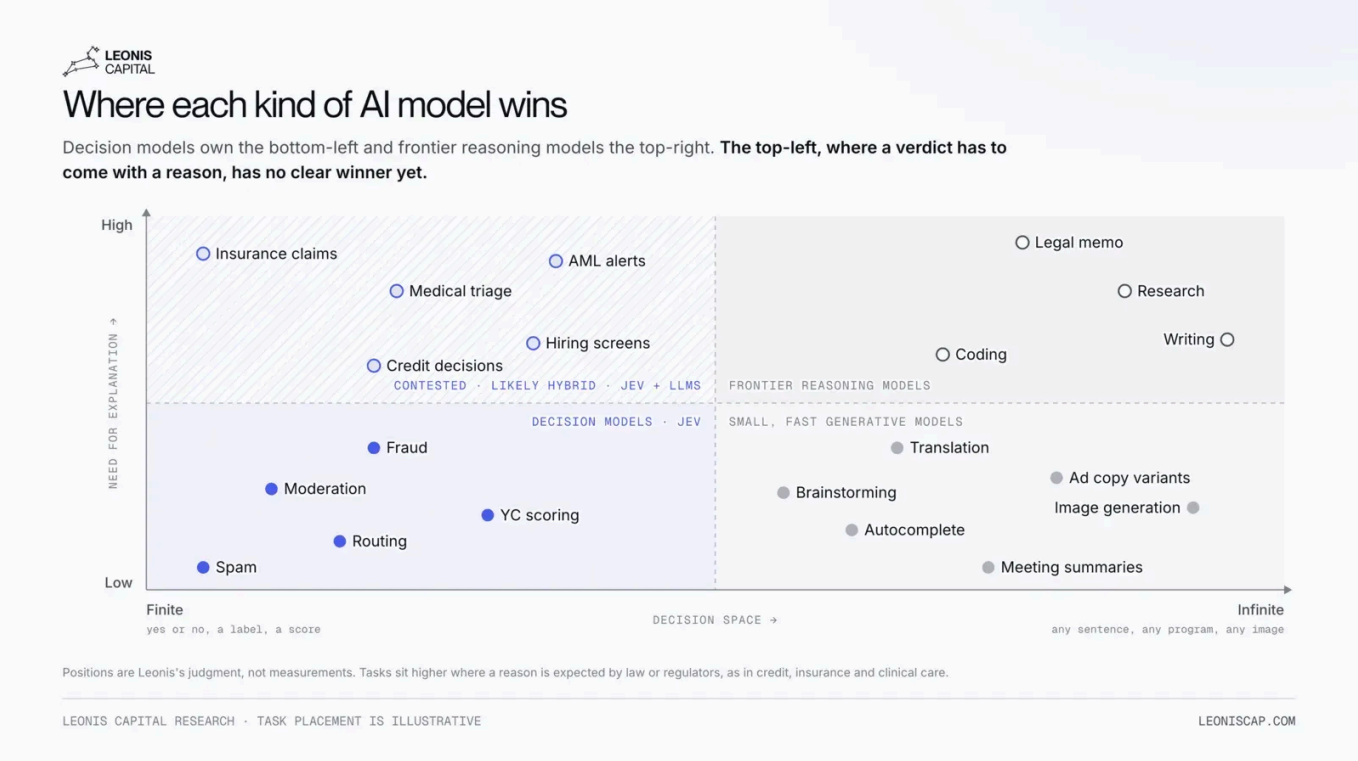
Finally, specialization cuts both ways and the obvious tradeoff is that **Jev cannot generate**. Jev can decide whether an email is phishing, but it cannot draft the warning to the user. Anything requiring writing, explanation, synthesis, or open-ended reasoning still needs a generative model.

In practice, Jev is more likely to sit alongside generative models than replace them in a flow where Jev decides; an LLM explains, writes, or reasons when needed.

What Cheap, Fast Decisions Unlock

What we call intelligence inside an application is often a series of distinct jobs-to-be-done. General-purpose LLMs made it possible to assign much of that work to one model. But as inference becomes a meaningful part of application cost and latency, developers have an incentive to ask which jobs actually require a frontier model.

Specialized models like Jev do not need to replace an LLM to matter. They only need to specialize and perform particular jobs better, faster, or more cheaply.



A useful way to think about where decision models fit is along two dimensions: **how bounded the output space is**, and **how much explanation the task requires.** Jev is strongest in the bottom-left for tasks like spam detection, routing, fraud scoring, and our YC screener, where the possible answers are constrained, and software mostly needs the decision itself. Frontier reasoning models dominate the opposite corner, where outputs are open-ended, and **the reasoning is part of the product**. The more interesting territory is the top-left for bounded decisions where the verdict must also be justified, such as insurance, credit, or clinical triage. We expect many of these workflows to become hybrid where decision models make the fast initial judgment, while frontier models or humans supply deeper reasoning when needed.

For example, in insurance, a decision model could sort incoming claims by apparent complexity and flag missing information. An LLM could examine the supporting documents, while a claims specialist reviews consequential or disputed cases. Conventional code would enforce approval requirements throughout, with escalation determined by both uncertainty and the consequences of an error.

At that scale, small differences in latency and cost compound quickly. They can determine whether an architecture is viable when that classification sits inside every transaction, message, or agent step.

Fast and cheap decisions open up at least three distinct application opportunities:

- **High-decision-density workflows:** Decision models are most valuable where software has to make hundreds or thousands of bounded judgments continuously. A security platform may classify every event by severity, a support system may route every ticket, and a CRM may reprioritize every lead as new signals arrive. When the same judgment sits inside every transaction, message, or agent step, lower latency and cost compound quickly.
- **Always-on AI:** Most AI products today wait to be invoked. Cheap, sub-second decisions make it practical for AI to evaluate every message, transaction, update, browser action, or system event as it happens. While OpenClaw-style heartbeats were sometimes prohibitively expensive with LLMs, decision models make that always-on layer economically viable.
- **Probabilistic software:** Much of enterprise software still encodes judgment through brittle rule trees. Decision models make it possible to replace some of that hand-written logic with learned, probabilistic judgments: how suspicious is this transaction, how likely is this claim to require review, how qualified is this lead? Hard rules still matter, but more of the fuzzy judgment between them can move into models.

For builders, product design should also now focus on the decision tree and schema. Which choices should the system offer? What evidence should permit an automatic action? How will the product learn whether that action was correct? A lead-scoring application, for example, needs to connect its scores with subsequent sales outcomes. Otherwise, it can produce confident ratings indefinitely without knowing whether they are effective.

As decision models become more widely adopted, companies with relevant historical data, verified outcomes, and established workflows can use their proprietary data to refine the questions, thresholds, and review paths around the model. Access to inexpensive intelligence makes building easier, shifting the product work to knowing when/how to apply and evaluate it.

These opportunities are more important than simply improving gross margins on existing AI products. Lower latency and cost can change what kinds of products are practical in the first place. Workflows that require thousands of judgments, or decisions fast enough to respond as events unfold, become more feasible as the models performing those jobs become more efficient.

At the model layer, specialization creates competition for individual jobs within an application. General-purpose LLMs will need to justify their higher cost where builders can choose a capable specialist. As specialized models improve and compete on price, application developers gain more affordable ways to build products that require frequent, rapid decisions.

Jev is an early example of what happens when model design starts with the needs of software. Its broader significance will depend on what developers can build once they no longer have to send every job through a general-purpose LLM.