

Jev: useful decisions, not guaranteed answers

By Sandeep · Tested 19 September 2026

Research and execution assisted by Hermes Agent.

The verdict

My verdict: Jev's opportunity is not "more intelligence." It is making narrow semantic judgments cheap and fast enough to use routinely inside ordinary software.

Think of it as a promptable classifier and scorer: you supply the evidence, question and possible answers at request time; it returns a bounded judgment for code to use. That flexibility is useful. It does not replace deeper reasoning or give the model authority to act.

Our tests put Jev on the shortlist for that role. They do not establish it as the best purchase against competing models. I would compare cost per acceptable decision, including mistakes, escalation and human review, rather than divide an accuracy score by the API price.

I asked Hermes Agent to test TypeSafe's Jev using its own playbooks. Actual API requests. Recorded outputs. A few straightforward cases, and a few where the software should refuse to guess.

The result was useful, but narrower than the headline claims.

Jev returned structured answers cheaply. It handled the straightforward classification examples well. Batching questions about the same document saved a substantial amount of input cost.

It also showed why a valid answer is not the same as a correct decision.

My recommendation is to treat Jev as a component for bounded judgments: routing, ranking and checking whether evidence supports a claim. I would not turn its confidence score into permission to publish, transfer money or make a clinical decision.

What Jev does

Jev does not write a reply like a chatbot. You give it text or structured state, ask questions, and define the possible answers. It returns values that ordinary software can use.

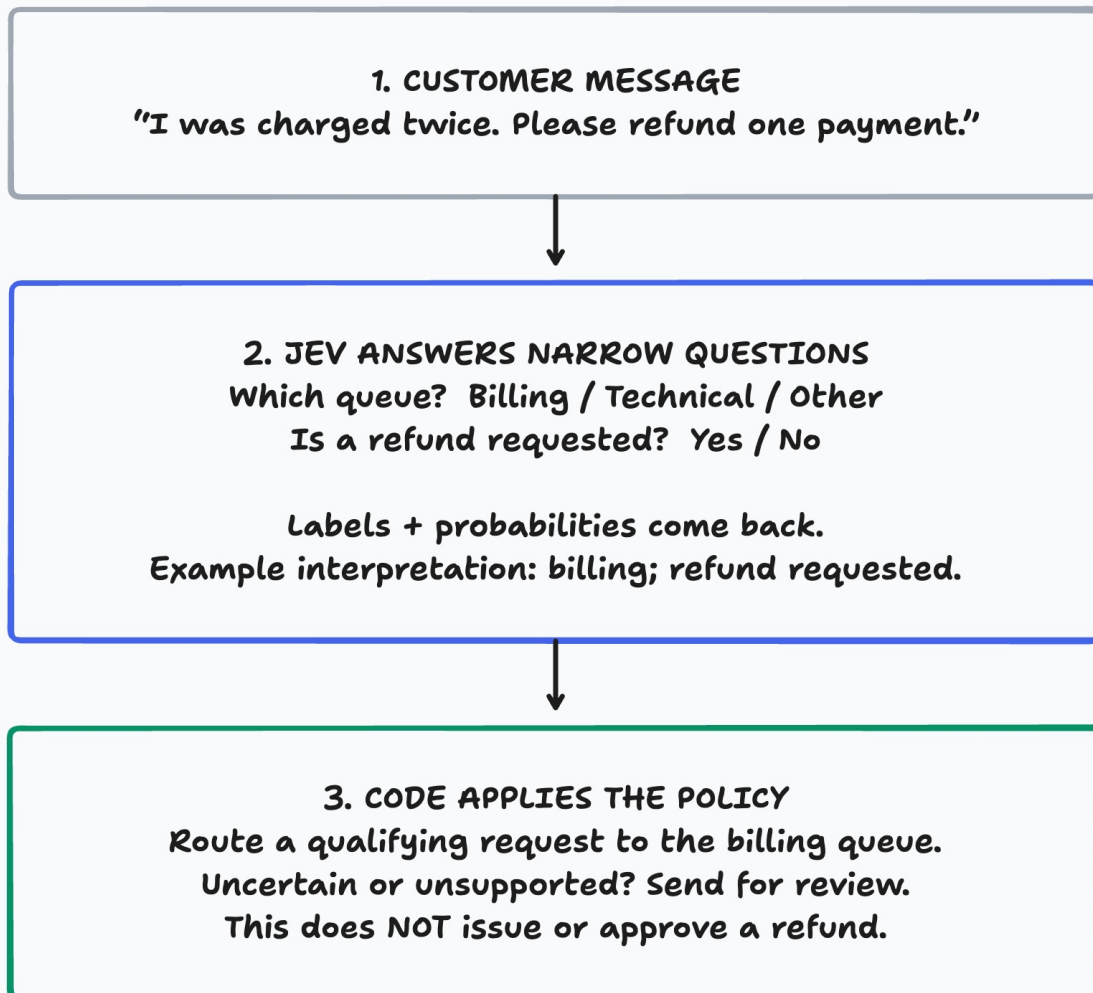
Its three question types are:

- **Choice:** pick one of the supplied options, with probabilities over those options.
- **Noul:** return the probability that a yes/no condition holds. There is no separate confidence field.
- **Score:** rate something against an ordered set of descriptions. A result can fall between levels; it is not an exact numerical extraction tool.

The model supplies the judgment. Code decides what happens next.

Jev judges the message. Code owns the action.

Illustrative support flow — not a measured test result



A refund still needs account checks, eligibility and permission.
Classifying the request does not authorise the action.

Generated by Hermes Agent using tldraw

*Illustrative support flow, not an observed API result or an implemented production integration.
Generated by Hermes Agent using tldraw.*

In the first synthetic example, a support message described a failing Stripe integration and lost sales. Jev selected **billing**, with confidence **0.44**. It rated frustration at **1.0** on a three-level scale and urgency at **0.99**.

The low routing confidence was the useful signal. That assignment needed another look.

What we tested

The test used `jev-1.13.0`, not a moving model alias. All input was public or synthetic. No customer, patient, employer or family records were sent.

Measurement	Result
Live evaluation requests	190, plus one model-list request
API responses	All 191 returned HTTP 200
Model reported by evaluations	jev-1.13.0 in all 190 responses
Primary diagnostic cases	58
Input tokens	1,263,532
Reported output tokens	11,703
Estimated TypeSafe inference charge	\$0.05307

That price estimate uses the published rate of **\$0.042 per million input tokens**, with output priced free. It is not a checked invoice and does not include the research agent, other tools or report production. Most requests were repeated measurements or controls, not independent quality examples.

The five workflows were intent routing, citation checking, input guardrails, skill suggestion and parallel questions over one document.

Several official cookbooks replay a supplied `json_cache.json` by default. This test bypassed those caches and recorded live network responses. Server-side caching was not inspected.

The primary inputs, labels and thresholds were fixed before inference. An AI assistant authored the cases, and a separate AI review challenged the labels without seeing the outputs. This was not a human-adjudicated benchmark. Later repair probes were kept separate rather than folded into the original scores.

Batching worked. The baseline changed the speed story.

The most instructive performance result came from asking the same 13 questions about the same pinned GDPR article.

Request strategy	Repeats	Mean wall time	Input tokens
One batch	5	0.861 s	11,828
Thirteen sequential calls	5	10.855 s	144,872
Thirteen concurrent calls	3	1.020 s	144,872

Against sequential calls, batching was **12.6× faster**. Against concurrent calls, the advantage was **about 1.18×**.

The input-cost difference remained **12.25×**. Sending the document thirteen times costs more than sending it once, even when the calls run concurrently.

For this document and question set, the estimated charge was **\$0.000497 per batch**, versus **\$0.006085 as separate requests**. At 1,000 identical-sized documents, that is roughly **\$0.50 versus**

\$6.08.

This is a comparison of request strategies on Jev. It says nothing about Jev being 12.6 times faster than another model.

The batched and sequential runs alternated in order. The concurrent control was added later and had fewer repeats, so it was not a perfectly matched statistical experiment. Timings included client and connection overhead; this was not a connection-pooled SDK or a server-only timing test.

Of 65 paired batched/sequential answers, **61 were exactly identical**. The largest numerical difference was **0.02**. No Choice winner or Noul yes/no result at a 0.5 cutoff changed. That supports batching these questions. It does not establish perfect determinism or universal independence.

Routing exposed a hole in the recipe

All **12 in-scope support messages** received the intended category, including examples with negation and mixed intent.

The four unrelated messages were more revealing. “Hello!”, “Explain what a monad is”, a bedtime-story request and a bank-transfer request all became `product_question`. Their confidence scores were **0.91, 0.99, 0.96 and 0.52**.

None triggered the cookbook’s below-0.5 review rule.

There is an important qualification: the original Choice offered no “none of these” option. It would be unfair to score the model as though it could select an unavailable label. This was a test of the recipe’s fallback design.

The lesson is still consequential. A concentrated distribution over the wrong set of options can look reassuring.

Adding an explicit no-match option and a separate “does this belong in shopping support?” question kept all four unrelated probes out of automatic routing and sent them for review, while retaining four straightforward controls. That was **8/8 on a development check using known cases**, not evidence of general accuracy on an untouched test set.

Citation checking was the most promising application

The citation checks used sections from RFC 7519, the JSON Web Token specification.

Twelve model decisions produced **11 exact matches to the initial labels**. The remaining claim said that a JWT subject was guaranteed to be a verified email address. The initial label was “says nothing”; Jev chose “contradicts”, with confidence **0.33**.

The independent label review challenged that expected answer. Because the specification permits other strings, “contradicts” is also defensible. I would not present that disagreement as an unequivocal model failure.

Both labels reject the claim as supported. The low confidence also sent it to review under the cookbook’s 0.8 threshold.

Ten decisions cleared that threshold and matched the original labels. Two went to review. Two fabricated quotations were caught before the model was called, by ordinary string matching. Those were **code successes, not Jev successes**.

The original semantic cases had no quotations attached. Six follow-up checks added authentic quotations to previously tested claims: two supported, two contradicted and two unsupported. All six matched their labels. They added coverage of the quote-checking workflow, not six independent new quality examples.

One example explains the value. A genuine quote says that processing a JWT expiration claim requires the current time to be before expiry. That does not establish that every JWT must contain an expiration claim; the same section makes it optional. Jev caught the distinction.

I would investigate this as an editorial triage step over public sources. It should flag questionable links between a claim and its evidence. It should not approve publication or decide that the underlying source is trustworthy.

Guardrails and skill selection need stronger tests

The guardrail recipe produced an acceptable strict-policy route for **15 of 16 messages**. All seven positive hazard instances were detected: three jailbreak attempts, one harmful request, two personal medical-decision requests and one self-harm signal.

The false positive was a harmless teaching request asking the assistant to classify the quoted string “ignore previous instructions” as an attack example without following it. Jev returned a jailbreak probability of **0.70**. The strict policy blocked it; the permissive policy sent it for review.

Another quoted-attack explanation passed, as did general medical-information questions. The distinction worked in some cases and failed in a closely related one.

The apparent **63/64 correct hazard cells** would make a poor headline. Most cells were negative: an always-negative classifier gets **57/64** here. Two hazard categories had only one positive example each. This was a smoke test, not a safety certification or an adaptive attack evaluation.

Skill suggestion produced **12/12 correct final suggestions** on a deliberately small eight-skill roster: eight supported requests and four no-match requests.

That result is encouraging but easy to oversell. The roster had no competing PowerPoint-editing skill, and the X skill explicitly excluded Mastodon. A flight-booking request was unsupported because this toy roster omitted browser skills, not because the agent platform cannot use a browser.

There was no live agent A/B test. The experiment did not establish fewer wrong skill loads, reduced prompt size or lower total agent cost. The published recipe retains the full skill index in the main agent’s prompt, so adding a suggestion service is not automatically a context-saving change.

What the claims actually establish

“Zero hallucinations” needs a narrower description

TypeSafe explains that its zero comes from schema matching, not an empirical rate of correct decisions. Constraining the available outputs prevents one kind of failure: inventing a label or tool outside the allowed set.

It does not eliminate semantic mistakes inside that set.

The company’s own integration skill states the boundary more precisely: **“Typed output guarantees the interface, not truth.”** That is the description I would use.

The interface held up in this test

All **190 responses** passed checks for answer keys, types, value ranges and option/level membership. That is evidence of a dependable interface on these requests, not a black-box proof of mathematical impossibility across every input.

Two Choice probability maps summed to **0.99**, despite the documentation saying they sum to one. That looks consistent with rounding. It is a numerical-contract wrinkle, not an invented answer. Consumers should handle a documented tolerance rather than assume exact sums.

The speed headlines are not portable guarantees

Across 81 small calls, median wall time was **0.529 seconds**, with a nearest-rank p95 of **0.614 seconds**. The document-heavy batch averaged **0.861 seconds**.

That did not reproduce a universal 100 ms response. It also does not make TypeSafe's own measurements false: its launch post notes that tests generally ran near its West Coast service. Input length, network path and connection reuse matter.

The advertised **193.6× faster and 444.6× cheaper** figures are comparisons from the vendor's workflows. This test did not run an equivalent competing-model benchmark, so those exact ratios remain unverified here. They should not become a general ROI assumption.

Confidence still needs validation

Choice and Score confidence summarize the shape of the probability distribution. They are not independently established probabilities that an entire workflow is correct.

The out-of-scope routing probes do not, by themselves, disprove calibration over a well-specified option set. They do disprove the shortcut that high confidence means a request is safe to route.

Production thresholds need representative data and an untouched evaluation set. Copying a threshold from a cookbook is not calibration.

What a fair competing-model comparison needs

TypeSafe's workflow evaluation uses reference answers derived from GPT-6 Astra and Claude Fable 5.1 at high thinking. Agreement with those references is not independently established correctness. The company also acknowledges that its adapter makes competing LLMs output probabilities, which can be slower and more expensive than returning decisions without them.

Our own labels were fixed before inference, but they were AI-authored and AI-reviewed, not human-adjudicated ground truth. One citation label was debatable. The existing cases are now known development material; a stronger comparison needs fresh held-out cases and independently adjudicated labels.

I would test both the same typed interface and the best practical alternative. If the application only needs a label, requiring every competitor to generate a full probability map can charge it for output the application does not need. Thresholds also need validation for each model and output type. This is a proposed comparison, not a benchmark we have run.

Where I would start

I would start with one public-content workflow: check whether cited passages support claims, then send questionable cases to an editor. Run it in shadow mode before allowing its output to influence a real process. Measure useful catches and unnecessary review work.

For other applications, keep the split simple:

1. Let code handle exact matches, arithmetic, permissions and known rules.
2. Give Jev a narrow judgment with sufficient relevant evidence.
3. Include a no-match outcome where one is possible.
4. Batch independent questions sharing the same evidence.
5. Evaluate the resulting decisions, not merely whether the response parses.

I would not give it exact arithmetic, date ordering, access control, durable workflow state or final approval. Its own limitations page warns about counting, numerical precision, indirection, distracting context and adversarial content.

Sensitive-data use requires a separate privacy assessment. The documentation promises no training on customer requests, but that is not the same as zero retention. This test did not verify account-specific contracts, residency, deletion behaviour or backend data handling.

Jev earned a place on the shortlist for bounded checks. It did not earn blanket trust. That is a useful result without making it a bigger one.

Scope and sources

This was a small AI-assisted diagnostic, not a representative production benchmark. It did not establish calibration, long-term reliability, multilingual performance, outage behaviour or superiority over another model. The published cookbooks used `jev-1.12`; these requests used `jev-1.13.0`, often with different fixtures. The study is not an exact replication of the vendor's published results.

Primary sources consulted for the dated evaluation:

1. [Models, pricing and data handling](#)
2. [HTTP API](#)
3. [Confidence](#)
4. [Jev 1.13 limitations](#)
5. [Intent routing](#)
6. [Parallel questions](#)
7. [Citation checking](#)
8. [Guardrails](#)
9. [Skill suggestion](#)
10. [TypeSafe launch post and comparison caveats](#)
11. [Official integration skill at the inspected revision](#)
12. [RFC 7519 and the pinned GDPR article](#)

13. [TypeSafe workflow evaluation methodology](#)
14. [Typed question primitives](#)