From Jev's probabilistic decisions to a deterministic Lean kernel

Matha Labs Research | November 15, 2026 | Prototype 0.1

Abstract

Typed neural decisions and formal verification fit different parts of an automation system. A learned model can interpret messy language and propose a bounded answer. A formal program can check whether fixed evidence and explicit rules permit that answer. This paper describes a small executable bridge between those roles, motivated by TypeSafe AI's Jev and Lexifina's use of Lean for legal authority. The implementation combines a TypeSafe-shaped response adapter with a Lean kernel, a fixed reviewed source, probability gates, and a versioned policy. It includes 12 named theorems, 24 Lean regression examples, and 18 adapter tests. All evaluated inputs are synthetic. The result is a reproducible demonstration of deterministic evaluation after a probabilistic proposal, with the remaining semantic trust boundary stated explicitly.

1. What Jev is

Jev is the actual TypeSafe AI model referenced in this project. TypeSafe introduced it on September 15, 2026 as its first System One model. The public launch describes a purpose-built architecture, parallel output evaluation, and a training approach called Reinforcement Learning for Calibrated Decisions, or RLCD. Those disclosures explain the intended interface and objective. They do not provide enough information to reproduce Jev's weights, architecture, training data, or complete training procedure. This project implements no replacement neural model. TypeSafe launch

The application supplies shared state and narrow typed questions. Choice selects among declared alternatives; Score evaluates an ordered rubric; Noul gives a probability that a yes/no statement is true. Current model documentation also lists the image-oriented Box primitive for Jev 1.14. The useful design idea is that ordinary application code retains control over which decisions exist and what follows each answer. Introduction, Models

These outputs remain judgments under uncertainty. Calibration describes how groups of predicted probabilities correspond to observed outcomes; it cannot certify an individual case. TypeSafe's own consistency experiments measure remaining variation when the same request is repeated. A deterministic downstream checker does not remove that variation from the neural model. System One, Self-consistency experiment

2. A precise meaning of deterministic results

Three properties should be evaluated separately. Replay determinism means the same input packet and rule bundle produce the same output. Logical validity means an accepted output satisfies the formal predicates encoded in the program. Real-world correctness means the underlying facts, interpretation, and applicable policy match reality.

The bridge implements the first property and proves specific statements about the second. It cannot obtain the third merely by compiling. If a source is false or a reviewer misinterprets a sentence, a perfectly deterministic

kernel can repeat the same error. The engineering task is to make those assumptions visible, narrow them, and attach enough provenance for independent review.

Two constructions follow. A fully symbolic decision model accepts already structured facts and computes a deterministic answer. A hybrid decision model accepts a neural proposal, binds it to reviewed facts, then runs a symbolic policy. The implementation here is the second construction. Calling it a newly trained Jev model would misdescribe what was built.

3. The relationship to Lexifina

Lexifina's November 13 article provides the motivating separation: software and language models extract claims, while an encoded authority model checks certificates and legal relationships. Its article describes tying claims to source sentences and checking whether a court possessed the authority attributed to it. Its reported corpus, measured results, and theorem suite belong to Lexifina; this prototype reproduces none of those experiments. Lexifina article

The transferable principle is that a model's answer should arrive with inspectable evidence and pass rules independent of its confidence. For clinical coding that could mean a documented finding plus a scoped coding rule. For reimbursement it could mean an effective policy, documented service, and explicit arithmetic. For permitting it could mean jurisdiction, work scope, rule version, and required review. The bridge demonstrates this pattern with a smaller fictional refund record, allowing its assumptions and tests to be read in full.

4. Architecture and implemented scope

The data path is:

```
Fixed synthetic source + versioned questions
                    |
        TypeSafe-shaped probabilistic response
                    |
        Strict Python normalization boundary
                    |
          Integer JSON candidate packet
                    |
     Lean kernel + fixed reviewed policy bundle
                    |
            accepted / review / invalid
```

The source states that transaction TX42 was charged twice and that its duplicate amount was USD 25. The bundled policy records a reviewed fact about that sentence and a validity interval. Neural output cannot edit this policy or turn its reviewed-fact flag on. The adapter always constructs the same closed question set. The Lean executable checks the resulting packet against the local bundle.

Acceptance requires a recognized pinned model version, valid probability mass, the requested action, an exact source identifier and quote, the reviewed fact identifier, a true reviewed-fact flag, an applicable policy date, and all numerical gates. Missing fields fail parsing. Unsupported choices, wrong model names, and invalid probability totals produce `invalid`. A well-formed packet that lacks sufficient support produces `review`.

This implementation checks equality with a bundled source line. It does not run an entailment prover over arbitrary text. The review that associates the sentence with a formal fact remains external. Keeping that distinction explicit makes the prototype useful for studying stronger certificate formats later.

5. Probability handling

The bridge uses an illustrative success threshold of 0.98 for Choice and a separate support threshold of 0.98 for Noul. The Score rubric has levels zero, one, and two; the allowed expected-risk value is at most 0.05. These constants have no demonstrated calibration or operating suitability for healthcare, reimbursement, or permits.

Choice and Score return distributions. Noul returns a yes probability and has no separate confidence field. Score is an expected position on a declared rubric, so the adapter recomputes it from the probabilities before trusting its shape. Noul, Score

The adapter validates Choice/Score confidence as metadata but does not count it as independent support. TypeSafe explains that confidence is derived from distribution concentration; for a fixed Choice cardinality it is a transformation of the top probability. Requiring both would not provide two independent tests of correctness. Confidence

Floating-point ambiguity is avoided at the Lean boundary. Python parses decimal JSON exactly, checks finite ranges and probability mass, then converts to basis points. It places quantization remainder on the abstention option and highest risk level. The Lean kernel uses natural-number arithmetic to recheck totals and thresholds. This is deliberately conservative at the conversion boundary; it is not a theorem about the quality of the original model probabilities.

6. What is proved and tested

The named theorems characterize acceptance and prove that acceptance requires the reviewed fact, certificate, pinned model, current policy, Choice gate, Noul gate, and risk bound. Exact quote equality is part of the certificate predicate. Further theorems show that no probabilities can override a false reviewed fact or an invalid certificate, and that an unrecognized model is invalid. The replay theorem states that equal policy and packet inputs imply equal outputs. These are quantified statements about the encoded kernel, independent of the 24 specific regression examples.

The examples exercise positive acceptance and failures for wrong provenance, wrong facts, stale or premature policies, empty evidence, unsupported actions, moving aliases, threshold boundaries, malformed distributions, and maximal confidence attached to an unreviewed fact. The Python tests include actual process calls to the built Lean executable, covering accepted replay, uncertainty review, forged evidence review, and invalid normalized input. A source scan rejects proof holes and explicit project axioms.

Passing these checks establishes a defined executable contract. It does not establish domain completeness, clinical accuracy, legal validity, adversarial robustness, or Jev calibration. No latency, cost, or accuracy results are reported because no live model was evaluated for this release.

7. Toward a larger system

A practical extension would freeze source documents, identify effective rule versions, and review the mapping from source language to formal facts. A proposer could choose among code-owned source spans, following the useful pattern TypeSafe documents for selecting pre-parsed values. Formal evaluation would then operate on the chosen spans and authenticated facts. Pre-parsed extraction

For a learned Jev-like research model, one could collect typed domain questions with adjudicated answers, train a discriminative predictor, and evaluate calibration, selective risk, repeated-call stability, and

out-of-distribution behavior. A Lean verifier could supply structural constraints or verifiable rewards for formally expressible parts. That is a proposed research program, not work completed by this repository. Training on a verifier also would not make the verifier's assumptions automatically correct.

The immediate contribution is smaller and inspectable: an open implementation showing exactly where probabilistic judgment ends, where deterministic policy begins, and what must still be reviewed before either is trusted in a real workflow.

Reproducibility

The repository pins Lean 4.35.0 and requires no additional Lean libraries. The Python adapter uses the standard library. `lake build`, the executable replay, the 18 Python tests, and the proof-source check run offline. The optional live adapter calls the official TypeSafe endpoint only when explicitly requested with `--live`; no credential or API call is needed for the published synthetic evaluation. Full first-party links and boundaries appear in `docs/sources.md` and `docs/evaluation.md`.