FORMAL METHODS / DECISION SYSTEMS

Deterministic Policy Evaluation After Probabilistic Typed Decisions

An executable Lean bridge for evidence binding, explicit abstention and reproducible synthetic decisions

Revanth Matha · Matha Labs Research whitepaper 2.0 | Revised 16 November 2026

CENTRAL CONTRIBUTION

An executable policy/packet separation with explicit acceptance, review and invalid-input outcomes for one closed synthetic refund fixture.

This is an independent research verifier around a TypeSafe-shaped interface, pinned to the original Jev 1.14.0 and Lean 4.35.0 snapshot. It contains a reviewed fictional duplicate-charge record, no trained neural model and no live-model performance evaluation. Acceptance is a result of the encoded synthetic policy and does not execute a refund or certify any healthcare, legal, municipal or tax decision.

Original publication: 15 November 2026 at 3:12:52 PM PST. This edition expands the paper; it does not claim a new trained model or a new clinical, municipal or financial deployment.

Contents

Abstract 3

Contributions 3

1. Problem and research question 3

2. Jev's public contract and disclosure boundary 4

3. Determinism, validation and proof are different strategies 4

4. Architecture and the closed reviewed fixture 5

5. Formal definitions and acceptance semantics 6

6. The normalization boundary and numeric representation 7

7. Probability, confidence and the meaning of the risk gate 8

8. Proof obligations and what the test counts establish 8

9. Five recorded cases from the published executable 9

10. Connection to Lexifina's certificate approach 9

11. Reproducibility, provenance and execution limits 10

12. A prospective empirical study and next research steps 11

References 12

Abstract

Software that consumes a learned judgment needs to distinguish an answer that fits its schema from one that satisfies an applicable policy. TypeSafe AI's Jev makes the first boundary explicit through typed decisions and probabilities. This paper investigates the second: whether a small Lean program can make acceptance conditions precise and replayable while retaining uncertainty and provenance. We implement a fixed synthetic refund policy, a structured proposal packet, a Python adapter for Choice, Noul and Score responses, and a deterministic Lean evaluator returning accepted, review or invalid. The published source contains 12 named theorems, 24 compile-time regression examples and 18 Python adapter tests. Five additional recorded scenarios show acceptance, altered evidence, insufficient probability support, an unrecognized model identifier and excessive rubric risk. These are authored fixtures executed by the published program; they are not observed Jev predictions or an accuracy benchmark. The proofs establish consequences of the encoded predicates, including the impossibility of overriding a false reviewed-fact flag or an invalid evidence certificate through probabilities. They do not establish source authenticity, semantic entailment, policy completeness or real-world validity. We relate this separation to Lexifina's source-certificate approach, describe the adapter and execution trust boundaries, and specify a prospective empirical study that compares selective error, coverage and repeatability under frozen policies. The contribution is an inspectable verification boundary and reproducibility record, with the conditions needed for stronger claims stated explicitly.

Keywords: Lean 4, typed decisions, Jev, formal verification, selective prediction, evidence provenance, deterministic evaluation

Contributions

1. An executable policy/packet separation with explicit acceptance, review and invalid-input outcomes for one closed synthetic refund fixture.

2. Twelve named Lean theorems, 24 Lean regression examples and 18 Python tests covering the acceptance contract and the response-normalization boundary.

3. Five reproducible, prerecorded executions with downloadable packets, hashes and the exact published source reference, accompanied by a proposed empirical evaluation protocol.

1. Problem and research question

A typed answer solves an interface problem: software knows which fields and options to expect. Business applicability is a separate question. A well-formed refund recommendation may refer to another transaction, cite an altered amount or rely on an expired policy. A useful acceptance mechanism must expose those dependencies rather than infer correctness from the fact that a response can be parsed.

Our question is narrowly operational: given a fixed proposed interpretation, a fixed reviewed fact and a fixed policy, can the decision be computed reproducibly and can its mandatory conditions be proved? The answer is demonstrated for a small synthetic case. A second question, whether this boundary reduces errors in a useful real workflow, remains empirical. No result in this paper answers it.

The implementation treats interpretation as an input-producing activity and acceptance as a separate function. This permits inspection of a failed gate without requiring a new explanation from a model. It also makes disagreements locatable: the source may be wrong, the semantic review may be wrong, the formal policy may be incomplete, or the packet may fail a condition. Those failures require different remedies. The published artifact provides a compact environment in which each dependency can be examined directly. [12]

2. Jev's public contract and disclosure boundary

TypeSafe introduced Jev as a System One model on September 15, 2026. Its launch describes a purpose-built architecture, a parallel sampler and Reinforcement Learning for Calibrated Decisions, abbreviated RLCD. These are public descriptions of a design and objective. They do not disclose a complete reproducible architecture, training corpus, loss implementation or weight checkpoint. We therefore implement an independent surrounding verifier, with no claim to train or reconstruct Jev. [1]

The public interface supplies shared state plus named, narrow questions. Choice selects among declared options. Noul returns a probability of a yes answer. Score returns a position on a declared ordered rubric with a distribution over its levels. Questions can be evaluated together while the application decides how their answers compose. Our adapter uses these three primitives. Current model documentation also describes an image-oriented Box primitive; it is outside this implementation. [2], [5], [6]

The original bridge requests and accepts the explicit identifier jev-1.14.0. TypeSafe distinguishes versioned identifiers from moving aliases, and its response includes the model name used. This makes version recording useful for reproducibility. A matching string in our packet is nevertheless an input assertion, not a cryptographic attestation that a provider produced the packet. All published fixtures are authored locally. Their model field identifies the intended interface contract. [3], [7]

3. Determinism, validation and proof are different strategies

One can obtain deterministic behavior without a theorem prover: write a pure rule function over structured inputs. A schema validator can reject malformed fields, and ordinary tests can exercise important examples. Lean adds a language in which the function and propositions about that function coexist. Proof checking can establish that an accepted result necessarily satisfies a specified predicate for all values quantified by the theorem. The requirement being proved still needs human scrutiny. [9]

Constrained neural output and deterministic decoding address another layer. They constrain the production of a candidate; they do not establish that the candidate is supported by the case. Recording one response enables replay even when a future inference call changes. TypeSafe's repeatability cookbook explicitly studies answer variation separately from correctness. We use that distinction without adopting its measured results as evidence for our own workflow. [8]

A fully symbolic system may be preferable when reliable structured facts already exist. The hybrid architecture is useful when interpretation is difficult but some consequential acceptance conditions are explicit. This prototype explores that division, not a universal requirement to insert Lean into every application. Its strongest property is the inspectability of a small acceptance contract.

Table 1. Four properties requiring different evidence

Property	What can establish it	Remaining question
Schema conformity	Declared options and field validation	Does the selected option describe this case?
Replay determinism	A pure evaluator with equal policy and packet values	Will another neural call produce equal values?
Formal acceptance validity	A checked theorem relating acceptance to explicit predicates	Do the predicates capture every intended rule?
Real-world correctness	Reliable sources, correct interpretation and domain evaluation	How often do accepted cases remain wrong in practice?

Conceptual distinctions used throughout this paper. The implemented theorem establishes formal consequences, while the empirical question remains open. [8], [9], [12]

4. Architecture and the closed reviewed fixture

The executable's local policy identifies a fictional transaction, TX42, with the source sentence: TX42 was charged twice; the duplicate amount is USD 25.00. The bundle includes a fact identifier, a reviewed-fact Boolean, a synthetic validity window and probability thresholds. It is deliberately small enough that the entire policy can be inspected. The reviewed flag represents an author-supplied attestation; no semantic review algorithm produces it. [12]

A Packet contains the proposed action, source and fact identifiers, quote, model identifier and integer probability fields. The external adapter converts a TypeSafe-shaped response into this packet. It copies the fixed local certificate metadata, so the model does not generate the source string or set the reviewed flag. The normalized packet is then parsed by the Lean executable and evaluated against its bundled policy. [13], [15]

This separation limits what a proposal can control, but it does not authenticate the ledger or prove that duplicate charging actually occurred. It also is not an arbitrary-document evidence engine. The current executable reads a packet file and compares selected fields with local values. It fetches no remote documents, contacts no payer or authority and performs no financial action.

Figure 1. The implemented boundary from response to decision



The published evaluation uses authored responses and packets. The optional live adapter exists but supplied no measured data for this paper. Policy review and source authenticity are inputs to this path, not conclusions of it. [12], [13], [15]

5. Formal definitions and acceptance semantics

Let p denote a Policy and x a Packet. The predicate packetValid checks a recognized model identifier, a declared action, two complete probability-mass totals and the range of the Noul value. Probability fields are natural numbers. Totals of 10000 imply the component bounds for each distribution. The model identifier is an exact string check rather than an inference about the packet's origin. [12]

The predicate certificateValid requires equality of policy identifier, source identifier, quote and fact identifier, together with a nonempty local source line. The full acceptable predicate adds a true reviewed-fact flag, inclusive policy bounds, the refund action, two minimum probability gates and a maximum expected-rubric numerator. evaluate returns invalid if packetValid fails; otherwise it returns accepted when acceptable holds and review in every other case.

Dates are natural-number values interpreted as YYYYMMDD in the closed fixture. The kernel proves numeric interval conditions, not a general calendar-validity theorem. The fixed November dates are intelligible, but an expanded policy loader would require calendar checks and a defined effective-time convention. Likewise, a nonempty quote is only the implemented test; authenticity, context and semantic entailment require stronger evidence outside this predicate.

These distinctions matter for rejection semantics. A well-formed but unsupported packet reaches review. Missing or incorrectly typed JSON fields fail at the parsing boundary with exit code 2. That parser failure is not the same result as a review decision from the total evaluator. [15]

```
A(p,x) = V(x) AND C(p,x) AND R(p) AND D(p) AND G(p,x)
```

V is packetValid; C is certificateValid; R requires reviewedFact=true; D requires validFrom<=day<=validThrough; G requires the refund action and the three numerical gates. For a valid typed packet, evaluate returns accepted exactly when A holds. [12]

Table 2. Implemented gates and their evidence limits

Gate	Implemented check	Not established
Model and mass	Pinned identifier; declared action; integer mass totals	Provider authenticity or probability calibration
Certificate	Exact local identifier and quote equality	Truth or contextual sufficiency of the source sentence
Reviewed fact	External flag equals true	The correctness of the external review
Policy date	Inclusive natural-number interval	Complete calendar and legal applicability semantics
Numerical support	Choice and Noul >=9800; risk numerator <=500	An observed bound on accepted-case error

A summary of the pinned definitions, not a complete policy for real refunds. Thresholds are synthetic design constants. [12]

6. The normalization boundary and numeric representation

The Python adapter validates the response object, exact question identifiers and expected primitive types. It checks option keys, finite numeric ranges, the selected Choice against the highest probability, and the exact code-owned Score legend. It recomputes the expected Score instead of accepting a conflicting supplied value. Duplicate JSON keys, nonfinite values and booleans presented as probabilities are rejected. These are important interface checks but are not themselves formally verified. [13], [17]

Probability sums must be within 0.000001 of one before normalization. The adapter then rescales by that total, floors selected masses to basis points and assigns remaining mass to abstention for Choice or the highest risk level for Score. Its direction of rounding is conservative relative to the normalized distribution. The tolerance, rescaling and decimal arithmetic remain part of the trusted adapter implementation; the Lean theorems concern the resulting integers.

Separating these layers avoids attributing a proof to the entire data path. The kernel independently rechecks relevant totals and gates after parsing. A correct integer packet can still encode the wrong semantic observation. Conversely, a legitimate provider response can be rejected by a deliberately strict adapter. Future interoperability evaluation should report both outcomes rather than silently weakening validation to increase apparent coverage. The existing 18 tests include representative adapter failures and selected calls to the executable. [17]

7. Probability, confidence and the meaning of the risk gate

The numerical gates express an application rule, not a statistical guarantee. A Choice threshold of 0.98 does not establish 98 percent accuracy among accepted cases. Acceptance also selects on correlated evidence, a second model judgment and the policy. Calibration in one population need not persist after such selection or under a different domain. The proposal and support questions share a source; the bridge makes no independence assumption and does not multiply their probabilities.

TypeSafe describes Choice and Score confidence as a statistic derived from the returned distribution. With fixed Choice cardinality, thresholding confidence and top probability adds no independent evidence. Noul has no separate confidence field. Accordingly, the adapter validates Choice/Score confidence as metadata while the kernel gates directly on normalized probabilities and the expected Score. Calibration remains a property to test across predictions, not a certificate for one case. [4], [6]

The variable called risk is the expected position in an illustrative conflict rubric: level 0 means no evident conflict, level 1 some ambiguity, and level 2 clear conflict. It is not a probability of injury, fraud or financial loss. Its allowed value is 0.05 on a 0-to-2 rubric. This scalar also loses distributional information: two distributions can have the same mean while allocating mass differently. A real decision policy may need tail constraints and a justified loss model. [5], [13]

```
E[level] = (risk1Bps + 2*risk2Bps)/10000 <= 0.05
```

The bridge computes the numerator with natural-number arithmetic and compares it with 500. The original accepted fixture has masses 9900, 90 and 10, giving (90+20)/10000=0.011. These values are authored examples, not measured risk estimates. [12]

8. Proof obligations and what the test counts establish

The 12 named theorems include accepted_iff; requirements for the reviewed fact, certificate, pinned model, current policy, Choice threshold, Noul threshold and risk bound; two impossibility results for bypassing a false fact flag or invalid certificate; invalidation of an unrecognized model; and replay determinism. Each statement quantifies over the encoded data types. Exact quote equality follows as a component of the required certificate. [12]

The bypass results are especially readable obligations. If p.reviewedFact=false, no packet can produce acceptance. If certificateValid fails, no probability values repair that failure. The replay theorem says equal policy values and equal packet values imply equal decisions. It is a basic consequence of the pure evaluator, not a claim that learned inference has become deterministic or that distinct JSON spellings necessarily normalize identically.

The 24 Lean examples cover concrete boundary cases, including threshold equality, altered provenance, date bounds, malformed mass, unsupported actions and a false fact flag paired with maximal probabilities. They supplement the quantified proofs by making intended behavior easy to inspect. The Python suite has 18 tests. A source scan rejects explicit project proof holes and axiom declarations, but neither that scan nor a theorem count proves policy completeness. Lean's own foundations and the distinction between proof checking and compiled execution remain relevant trust boundaries. [10], [14], [17]

9. Five recorded cases from the published executable

The article's interactive panel selects five saved executions, rather than running Lean in the browser. Each packet is tied to the exact code reference, has a SHA-256 digest and records the executable's output and exit code. Per-condition indicators are explanatory computations over the frozen fixture; the final decisions come from the actual Lean executable. The cases are a demonstration of mechanisms, not a sampled dataset from which accuracy or error prevalence can be estimated. [16]

The accepted case combines a 0.99 refund probability, 0.995 support probability and a 0.011 expected rubric score with a matching certificate. The altered-quote case preserves those numbers but changes USD 25.00 to USD 250.00 in the proposed quote; review follows from the mismatch. Weak Choice lowers the refund mass to 0.97 while preserving total mass, producing review even though the certificate still matches.

The unpinned-model case substitutes jev-latest for the required identifier. That alias is recognized by the provider's API, but it is unrecognized by this particular packet contract, so the recorded result is invalid with exit code 2. Finally, the elevated-risk distribution places 94 percent on level 0, 4 percent on level 1 and 2 percent on level 2. Its expected value is 0.08, above the fixture limit, and the result is review. No inference call produced any of these probabilities. [3], [16]

Table 3. Mechanism checks with one intended change per case

Case	Changed input	Recorded result	Explanatory failed gate
Supported refund	None; original synthetic packet	accepted; exit 0	None
Altered evidence	Quote amount 25.00 becomes 250.00	review; exit 0	Exact certificate
Weak Choice	Refund mass 99% becomes 97%	review; exit 0	Choice minimum 98%
Unpinned model	Model field becomes jev-latest	invalid; exit 2	Pinned model identifier
Elevated risk	Rubric expectation 0.011 becomes 0.08	review; exit 0	Maximum expectation 0.05

Recorded outputs from the five downloadable packets. Their engineered differences exercise independent gate requirements; the word independent here describes separate rules, not statistical independence of model predictions. [16]

10. Connection to Lexifina's certificate approach

Lexifina's supplied case-law article describes extracted claims carrying source certificates and Lean checking encoded conditions concerning authority, court relationships and dates. The relevant methodological separation is between reading a text and establishing that a formal rule licenses a conclusion about it. The article also reports its own corpus and validation work. We have not reproduced those experiments, and none of its measured performance transfers to this bridge. [11]

Our certificate is substantially weaker and smaller: equality against one embedded sentence and known identifiers, plus an externally set reviewed flag. It does not identify speakers in an opinion, interpret judicial authority or prove that natural-language text entails a structured fact. That difference is essential. A citation that points somewhere is not yet an entailment certificate, and an entailment certificate would still depend on the chosen formal semantics.

One practical extension would let code enumerate candidate source spans and allow a model to select among them, while retaining offsets and document hashes. TypeSafe documents a comparable division for selecting pre-parsed values. This could prevent invented strings while leaving the harder question, whether the selected passage supports the claim in context, for independent review or a separately evaluated procedure. The current bridge supplies a controlled starting point for that extension. [18]

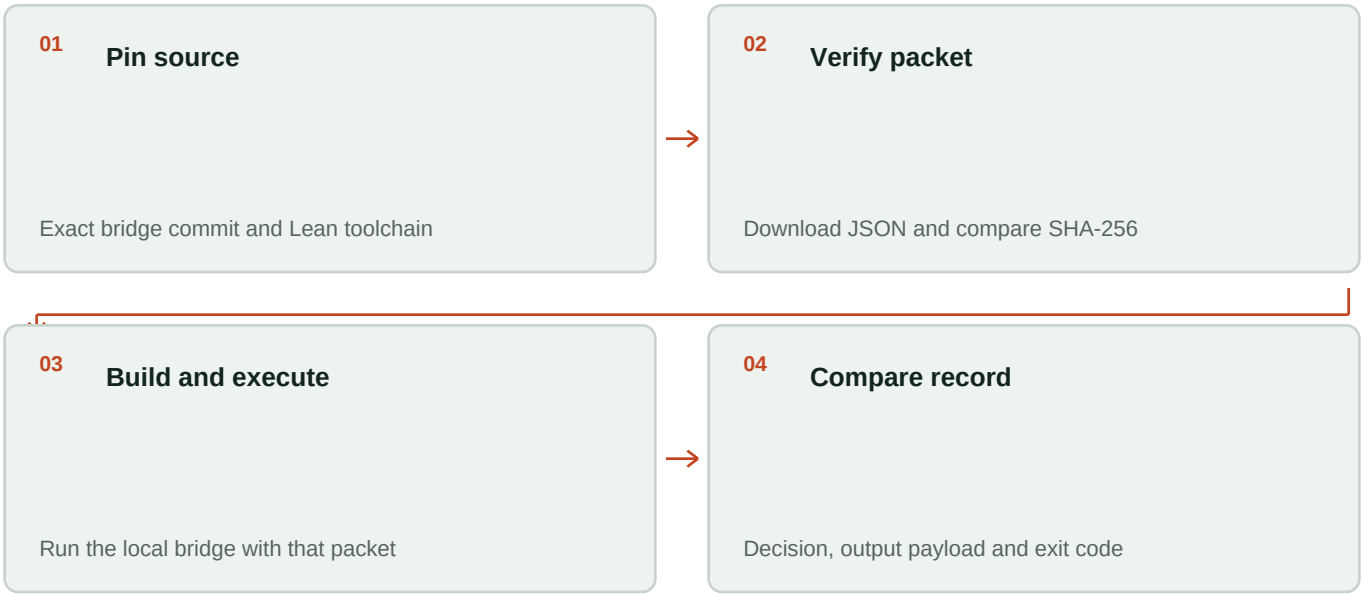
11. Reproducibility, provenance and execution limits

Reproduction begins with code reference 2b8b8af1d0322d449f08f6492fd6476632c07a92 and lean-toolchain specifying Lean 4.35.0. The project uses the Lean distribution's libraries and the Python standard library. Build with lake build; run lake exe bridge with a downloaded packet; then execute the 18 Python tests and the proof-source scan. Invalid packets intentionally return a nonzero exit code. The optional --live adapter is separate, requires credentials and was not used for the published evaluation. [12], [13], [15], [17]

Provenance has several layers. A code commit identifies source text; a toolchain version identifies a build dependency; a packet digest identifies bytes; and a transcript records observed execution. None alone establishes all the others. The article's manifest links these artifacts, but a reader seeking stronger assurance should rebuild from the source and compare outputs. Operating-system and compiler execution are not proved correct by the acceptance theorems. [10], [16]

Known limits include the closed fact catalogue, the absence of source authentication, natural-number date bounds, trusted normalization and parsing code, and no adversarial or domain-scale evaluation. The fixed reviewed-fact flag also creates a possible oracle effect: if expert review has already settled the important semantic question, the marginal contribution of a model may be small. An empirical design must separate the work done by reviewers, the model and the formal gates.

Figure 2. Reproduce an execution without a model account



A reproduction procedure for the existing synthetic artifact. It checks the saved decision path and makes no new Jev request. Hashes identify artifacts; they do not independently authenticate the underlying fictional record. [15], [16]

12. A prospective empirical study and next research steps

A useful next study should freeze one domain and a narrow decision task before gathering results. Collect a permitted corpus with independent adjudication, document the source of each fact and split related cases together to prevent leakage. Reviewers should label both the intended decision and the evidence supporting it. Develop the rule encoding and thresholds on separate development material, then lock the code, prompts, model version, source revisions and test protocol. This is a proposed procedure; no such corpus or evaluation is claimed here.

Compare a probability-only proposal rule, schema validation plus ordinary rules, and the same encoded rules implemented in Lean. Holding the rules constant helps distinguish the effect of an acceptance boundary from the benefit of proof tooling. Ablations can remove certificate binding or probability gates to identify their contribution. Measure error among accepted cases together with coverage and review workload; a system that refers everything can reduce accepted-case error without providing useful automation.

Report extraction failures, incorrect reviewed facts, policy-encoding errors, invalid packets and decision errors separately. Estimate uncertainty with methods that respect clustering by source or case. Evaluate repeated identical requests and wording changes separately from replay of a saved packet. Assess calibration before and after selection, and describe unsupported inputs explicitly. Larger samples are needed when the tolerated accepted-case error is small; this paper does not prescribe an unsupported sample size or target performance.

The immediate extension should strengthen provenance and domain evidence before increasing catalogue size. A larger theorem count cannot compensate for a poorly specified requirement. The present result is a runnable boundary: a model-shaped proposal can be inspected, rejected, reviewed or accepted under fixed formal conditions. Whether that boundary earns its complexity in practice is the next question to measure.

Table 4. Proposed comparisons and evidence to collect

Comparison	Hold fixed	Measure
Proposal rule versus formal gates	Cases, saved responses and decision labels	Accepted-case error, coverage and review workload
Ordinary rules versus Lean encoding	The intended rule specification	Differential disagreement, maintenance errors and proof obligations
Certificate and gate ablations	Model responses and test cases	Which errors each requirement excludes
Repeated inference versus saved replay	Request, model version and policy	Response variation, decision variation and exact replay agreement

Prospective study design. No row reports an experiment already performed, and no performance improvement is assumed.

References

[1] Introducing System One Models & Jev. TypeSafe AI, September 15, 2026. Public descriptions of the model architecture direction, parallel sampler and RLCD; vendor claims, not a reproducible training specification. Accessed 2026-11-16.

[2] Introduction - TypeSafe. Typed state/question interface, primitives and independent composition. Used for interface description only. Accessed 2026-11-16.

[3] Models - TypeSafe. Jev 1.14.0 model identifier, primitive support and moving-alias semantics. The implementation retains its original pinned snapshot. Accessed 2026-11-16.

[4] Confidence - TypeSafe. Distribution-derived confidence, its distinction from probability and the fixed-cardinality Choice relationship. Accessed 2026-11-16.

[5] Score - TypeSafe. Ordered rubric levels, probability-weighted expected score and distributional interpretation. Accessed 2026-11-16.

[6] Noul - TypeSafe. Probability of a yes answer; no separate confidence field. Accessed 2026-11-16.

[7] System One - TypeSafe HTTP API. POST /v1/systemone request/response structure and returned model identifier. The paper reports no live requests. Accessed 2026-11-16.

[8] Self-consistency: choices - TypeSafe. A vendor experiment separating repeated-answer stability from correctness. No vendor benchmark number is adopted as a bridge result. Accessed 2026-11-16.

[9] Theorem Proving in Lean 4: Propositions and Proofs. Propositions as types and the relationship between a theorem statement and its proof term. Accessed 2026-11-16.

[10] Theorem Proving in Lean 4: Axioms and Computation. Foundational assumptions and the distinction between logical content and compiled computation. Accessed 2026-11-16.

[11] Formalizing all US case law in Lean. Lexifina, November 13, 2026, especially sections 1 and 2. Source certificates and formal authority checks motivate the separation; its results are not reproduced here. Accessed 2026-11-16.

[12] Lean Jev Bridge: Bridge/Kernel.lean. Exact published source for Policy, Packet, acceptance predicates, evaluator, 12 named theorems and synthetic bundle. Accessed 2026-11-16.

[13] Lean Jev Bridge: scripts/adapter.py. Pinned Python response validation, normalization, basis-point conversion and optional live adapter. Accessed 2026-11-16.

[14] Lean Jev Bridge: Bridge/Tests.lean. The 24 authored compile-time regression examples; no clinical or real-world test corpus. Accessed 2026-11-16.

[15] Lean Jev Bridge: Main.lean. Normalized JSON parsing, bundled-policy evaluation, output payload and exit-code semantics. Accessed 2026-11-16.

[16] Matha Labs: recorded Jev-bridge executions. Five synthetic packets, pinned source reference, kernel/packet hashes, explanatory checks and saved executable transcripts. Accessed 2026-11-16.

[17] Lean Jev Bridge: scripts/test_adapter.py. The 18 Python adapter tests, including selected process calls to the Lean executable. Accessed 2026-11-16.

[18] Pre-parsed value extraction - TypeSafe. Code-owned candidate extraction followed by model selection; cited as a possible extension, not an implemented arbitrary-span verifier. Accessed 2026-11-16.

Reproducibility record

Source: https://github.com/RevanthM/lean-jev-bridge | Evaluated implementation: 2b8b8af1d0322d449f08f6492fd6476632c07a92