JEV ENGINEERING FOR PRODUCTION AGENTS

A Practical Handbook on Typed Semantic Decisions

Companion to 'Jev Engineering: Stop Using LLMs for Every Decision'

Independent study edition. Not affiliated with or endorsed by TypeSafe AI.

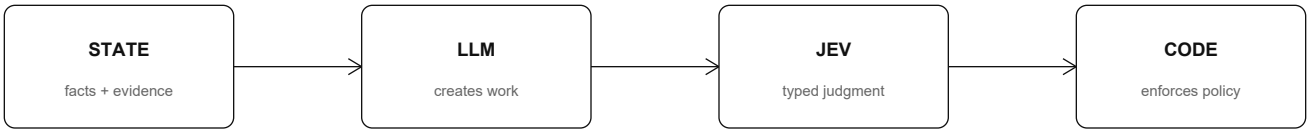

Fig. 1. Generation, semantic judgment, and authority are separate responsibilities.

ABSTRACT

Jev is a decision model, not a conversational model. It receives structured state together with typed questions and returns constrained answers plus probability distributions. This interface is useful inside agents because many expensive calls do not need new language. They need a route, a score, a yes-or-no estimate, or a decision to escalate. The generative model remains responsible for open-ended work; deterministic code remains responsible for exact invariants and external authority.

This handbook develops that division of labor as an engineering discipline. The central unit is the decision contract: a versioned specification of state, instructions, declared outcomes, fallbacks, thresholds, allowed actions, and escalation behavior. A contract turns a hidden judgment into production logic that can be evaluated, logged, reviewed, and rolled back.

The treatment is deliberately operational. It covers Choice, Score, and Noul; compact evidence packets; confidence-aware routing; parallel evaluation against one state snapshot; dynamic option menus; harness insertion points; shadow-mode evaluation; calibration; decision receipts; and failure modes that are often misdiagnosed as model defects. The goal is not to maximize Jev usage. The goal is to remove generative calls from decisions that never required a sentence.

TypeSafe reports approximately 70-500 ms end-to-end latency for Jev-shaped queries and a price of $0.042 per million input tokens with no metered output-token cost. Its broader 20-200x speed and 40-400x cost claims come from favorable workflow evaluations. Those figures should be read as workload-dependent ceilings rather than promises for every integration. The architectural value of typed judgment does not depend on reaching the headline maximum.

INDEX TERMS

Jev, TypeSafe AI, system one model, decision contract, typed judgment, calibration, confidence routing, agent harness, shadow mode, semantic verification, Choice, Score, Noul.

I.

WHY A DECISION LAYER

Decision Layer

A. THE EXPENSIVE ASSUMPTION

Most agent stacks assume that every intelligent judgment deserves another generative call. The loop asks the same frontier model to write, classify, select tools, estimate risk, decide whether evidence is sufficient, verify outcomes, and determine when to stop. From the outside this looks like a single capable agent. Internally it is a large collection of small branches implemented as prose generation.

The cost of those branches is easy to miss because it rarely appears in the final answer. It appears as sequential latency, repeated context, JSON repair, schema retries, and recovery after a weak branch. One additional call may be acceptable. Twenty dependent calls can dominate the workflow even when the visible artifact is short.

B. A DIFFERENT PRIMITIVE

Jev changes the primitive from text generation to typed semantic judgment. The caller provides current state and one or more predefined questions. The response contains declared answer types and probability distributions. Software does not need to recover a decision from prose, infer a confidence score from tone, or hope that an ad hoc JSON schema survives generation.

This restriction is the point. Jev does not draft the email, write the report, or produce the code patch. It can decide which queue should receive the email, whether the report has enough evidence, or whether the proposed patch should be reviewed before execution. Constrained outputs make those decisions composable with ordinary program logic.

C. LATENCY, COST, AND INTEGRATION

TypeSafe describes Jev as a system one model trained with Reinforcement Learning for Calibrated Decisions. The objective is decision quality together with useful uncertainty, not attractive prose. The reported service characteristics - approximately 70-500 ms for Jev-shaped requests and $0.042 per million input tokens without metered output tokens - make high-frequency semantic branching plausible.

Those service characteristics do not eliminate system design. A cheap decision that selects the wrong worker can create more downstream cost than an expensive correct decision. A fast verifier that passes incomplete work can add latency later. The unit of analysis is therefore the completed task, not the isolated call.

G. INVENTORY THE HIDDEN DECISIONS

Before integrating Jev, write down every semantic branch already present in the agent: request clarity, worker selection, model selection, retrieval survival, tool risk, completion, retry, escalation, and stop. Record which component makes each branch today and whether the branch creates language, interprets meaning, or enforces an exact rule.

The inventory exposes decisions that were previously buried inside prompts. It also prevents indiscriminate migration. Some branches should remain in the LLM because the output is generative; others should move into code because the condition is exact. Only the remaining semantic branches are Jev candidates.

D. THREE OWNERS INSIDE THE AGENT

A production agent should assign work to three different owners. The generative LLM creates open-ended artifacts. Jev interprets ambiguous meaning when the answer shape is known. Deterministic code enforces exact invariants. Mixing the three collapses distinct failure modes into one opaque model call.

QUESTION	OWNER	REASON
Draft a research summary	LLM	The output itself must be created
Choose the next worker	Jev	Meaning is fuzzy; the menu is known
Rate evidence quality	Jev	The rubric is ordered and semantic
Stop after three attempts	Code	The rule is exact
Approve a payment	Code + human	The side effect is consequential
Verify a report	Jev + code	A rubric and artifact checks are both needed

Table I. Workload ownership follows output shape and authority, not model capability alone.

E. AUTHORITY REMAINS OUTSIDE THE MODEL

A confidence score is an operating signal, not permission. Jev may estimate that a command is low risk, but the policy engine still decides whether it may execute. The distinction matters most for publishing, purchasing, deletion, account access, permission changes, database writes, money movement, and representation of the user.

The safe order is stable across integrations: Jev judges; code checks policy; the tool executes; the trace records. Reversing that order turns probabilistic judgment into unbounded authority. The classifier then contains the policy, and application code merely obeys it. That design is difficult to audit and unsafe to extend.

F. THE ENGINEERING OBJECTIVE

The objective is not to replace the LLM. It is to stop using generation where generation is unnecessary. Every removed prose call makes the graph smaller, the schema simpler, and the decision easier to evaluate. Jev engineering is therefore boundary design: deciding where language creates value, where typed judgment resolves ambiguity, and where code must remain non-negotiable.

G. AN OPERATIONAL BOUNDARY TEST

Ask three questions about a branch. Must the output itself be authored? If yes, use a generative model. Is the meaning ambiguous while the answer shape is already known? If yes, use Jev. Must the condition be obeyed exactly? If yes, use code. Branches that appear to need all three should be decomposed into generation, judgment, and enforcement steps.

This decomposition makes responsibility visible in both the implementation and the trace. A failure can then be assigned to artifact creation, semantic judgment, or policy enforcement rather than attributed vaguely to the agent.

II.

CHOICE, SCORE, AND NOUL

Decision Primitives

A. CHOICE

Choice applies when exactly one declared option should win. The important design work is not the option label but the option description. Each criterion must tell the model what evidence distinguishes that outcome from its neighbors. Labels such as research, write, and review are convenient for code; the descriptions make them meaningful.

Closed menus need an escape hatch whenever reality may contain an unlisted state. Without none, other, stop, or escalate, probability mass is forced onto the least-wrong option. The result remains type-valid but can be operationally false. An escape outcome preserves uncertainty and gives the harness a safe route.

```
Choice(
  instructions='Which worker should act next?',
  criteria={
    'research': 'important evidence is missing',
    'write': 'evidence supports drafting',
    'review': 'request is unclear or consequential',
    'none': 'no listed worker is appropriate'
  }
)
```

Choice is useful for worker routing, model selection, queue assignment, choosing one current interface control, selecting a retry strategy, and deciding which retrieval candidate should survive. It is not suitable for inventing an arbitrary identifier, URL, number, or name that was not declared in the menu.

B. SCORE

Score applies when the decision lives on an ordered semantic rubric. The levels should be verbal descriptions, not vague numbers. For evidence quality, the scale might move from no direct support to partial support with gaps to strong support from multiple independent sources.

Outputs may fall between rubric levels, but the interpolation does not convert an ordinal scale into exact measurement. A value of 1.6 is a position between two descriptions. It is not automatically 80 percent good, 1.6 times better, or equivalent across different contracts.

```
Score(criteria=[
  'no direct support',
  'partial support with important gaps',
  'strong support from independent sources'
])
```

G. RUBRIC DESIGN

Rubric levels should describe observable differences in evidence, not emotional intensity. Adjacent levels need a meaningful boundary. If reviewers cannot place examples consistently, Jev cannot be expected to learn a stable ordering from the descriptions alone.

Prefer three to five well-separated levels. More levels create an appearance of precision while reducing inter-rater agreement. Preserve the original level descriptions in the receipt so later analysis can distinguish model behavior from a contract change.

C. NOUL

Noul represents a yes-or-no probability. A result near one means likely yes. A result near zero means likely no. A result near one half means the available evidence does not support a confident distinction. It does not mean medium severity or partial permission. Ordered risk categories belong in Score.

```
Noul(
  instructions=(
    'Would the proposed action publish, purchase, '
    'delete, change permissions, or represent the '
    'user externally?'
  )
)
```

Noul is appropriate for completion checks, approval likelihood, evidence sufficiency, whether a request is ambiguous, and whether a proposed action has an external side effect. The application must still decide what probability range triggers automation, more evidence, or human review.

D. SELECTION DISCIPLINE

ANSWER SHAPE	PRIMITIVE	DESIGN REQUIREMENT
One winner	Choice	Describe each option; add an escape hatch
Ordered quality	Score	Define verbal anchors; avoid fake precision
Yes or no	Noul	Interpret uncertainty, not severity
Unknown free string	Not Jev	Use extraction or generation
Exact arithmetic	Not Jev	Use deterministic code

Table II. Primitive selection begins with the declared answer shape.

E. TYPE SAFETY IS NOT TRUTH

Jev can guarantee that the answer matches the declared output type. It cannot guarantee that the selected valid answer is semantically correct. Wrong evidence, stale options, ambiguous criteria, or an incomplete menu can produce a wrong choice that is perfectly valid at the type level.

Production systems therefore need representative examples, confidence thresholds, deterministic checks, decision receipts, and review paths. Type safety removes parsing and schema failures. It does not remove the need to test the meaning of the decision.

F. PRIMITIVE COMPOSITION

The primitives may be asked together when they inspect the same snapshot. A router can request next worker by Choice, urgency by Score, and approval likelihood by Noul in one call. They should not be chained inside the same request when a later question depends on new evidence created by an earlier answer.

G. PRIMITIVE ANTI-PATTERNS

Do not encode a multi-label problem as Choice if several options may be true simultaneously. Do not use Score when categories are unordered. Do not interpret a Noul probability as a severity scale. Each mismatch can produce plausible numbers that the application reads with the wrong semantics.

When the required output does not match one primitive cleanly, divide the question. A risk workflow might use Noul to detect the presence of an external side effect, Score to rate consequence, and Choice to select the permitted route.

III.

DECISION CONTRACTS AS CODE

Decision Contracts

A. A QUESTION IS PART OF THE PROGRAM

A production Jev question is a decision contract. It defines which state fields are visible, what the instructions mean, which outcomes exist, which escape path is safe, how confidence is interpreted, which action may follow, and which contract version produced the result. Treating the question as casual prompting discards the main advantage of typed judgment.

Internal identifiers are not instructions. A field named safe_to_publish helps application code but does not teach the model what safe means. The operational definition must appear in the instructions and criteria. Otherwise the system contains a semantic rule that is visible to developers but absent from the model input.

```
# weak
safe_to_publish = Noul('Is this safe?')

# stronger
safe_to_publish = Noul(
  'Does the draft contain only verified claims with '
  'citations, avoid private information, and require '
  'no unresolved legal or financial approval?'
)
```

B. CONTRACT FIELDS

A minimal contract includes: state schema; instructions; option descriptions or rubric; fallback outcome; confidence thresholds; consequence class; allowed action; escalation path; model version; and contract version. The contract should be reviewable without reading the entire application.

The allowed action field is especially important. It prevents a classification result from silently expanding into authority. A contract may allow internal routing but forbid any external side effect. The harness remains responsible for verifying that the resulting action stays inside that boundary.

C. CONTRACT VERSIONING

Options, users, tools, and language change. A semantic edit can alter production behavior even when no application code changes. Version the decision contract separately so that changes can be reviewed, evaluated against historical examples, deployed gradually, and rolled back.

D. REVIEW THE CONTRACT LIKE CODE

A contract review should verify that every state field is necessary, every outcome is distinguishable, every open menu has an escape hatch, and every confidence range maps to an explicit route. Reviewers should also confirm that the allowed action is narrower than the model judgment and that exact constraints remain in code.

Changes to criteria deserve regression tests. Re-run labeled examples from the previous version, inspect class-specific accuracy and calibration, and require an explanation for behavior changes. A shorter prompt is not automatically a safer or more maintainable contract.

D. THE DECISION RECEIPT

Every production decision should produce a receipt. The receipt links judgment to the state and policy that existed at the time. It is the minimum unit for auditing, calibration, incident review, and comparison between contract versions.

RECEIPT FIELD	PURPOSE
contract_version	Identifies the semantic program
state_reference	Links to the evaluated evidence snapshot
full_distribution	Preserves uncertainty, not only the winner
selected_threshold	Explains the operating zone
consequence_class	Explains why a route was permitted
route	Records auto, improve, or human
resulting_action	Connects judgment to execution

Table III. A label without a receipt is difficult to evaluate or audit.

E. CONTRACT LIFECYCLE

The lifecycle has four stages. First, define the state and outcomes. Second, evaluate the contract against one immutable snapshot. Third, route the result through consequence-aware thresholds and deterministic policy. Fourth, record the receipt and resulting action. No stage should be hidden inside another.

Separating evaluation from routing allows the same distribution to support different policies. An internal queue may automate at a lower threshold than a public statement. An irreversible action may require review at every confidence level. The semantic judgment is reusable; the authority policy remains context-specific.

F. TESTING CONTRACTS

Contract tests should include ordinary cases, ambiguous cases, missing evidence, adversarial language, stale options, and examples where no option fits. Tests should evaluate both the selected answer and whether confidence changes monotonically with evidence quality. A contract that is accurate only on easy examples is not ready to control a branch.

```
router@1  broad options
router@2  adds none-of-the-above
router@3  separates billing from account access
router@4  consequence-specific thresholds
```

G. RECEIPTS DURING INCIDENTS

During an incident, the receipt should answer what the model saw, which contract interpreted it, how probability was distributed, which threshold fired, which deterministic policy applied, and which action followed. Without those links, teams can observe the failure but cannot locate the faulty boundary.

Receipts also enable counterfactual evaluation. A new contract version can be replayed against historical state snapshots without repeating the original side effects. That comparison is essential when the goal is safer routing rather than merely different labels.

IV.

STATE PACKETS AND EVIDENCE

State Engineering

A. STATE ENGINEERING

Jev can only judge the state it receives. Sending a giant transcript forces the model to reconstruct the workflow from mixed instructions, old attempts, conclusions, and irrelevant history. The output may still satisfy the declared type while the underlying judgment is based on stale or misleading context.

A state packet should be compact, current, and evidence-based. Separate goal, facts, artifacts, evidence, constraints, options, and state version. Each field has a distinct function. The separation makes omissions visible and prevents an earlier agent's interpretation from becoming an unquestioned fact.

STATE FIELD	FUNCTION
Goal	Defines success for the current task
Facts	Records what the system currently knows
Artifacts	Lists outputs that already exist
Evidence	Supports the next semantic judgment
Constraints	Defines boundaries the system may not cross
Options	Enumerates what can happen now
Version	Identifies the exact evaluated snapshot

Table IV. Functional state fields reduce accidental coupling between evidence and interpretation.

B. EVIDENCE, NOT INHERITED CONFIDENCE

The state should contain observable evidence rather than conclusions. 'The research is probably enough' asks Jev to trust an earlier judgment. 'Seven sources collected, three official, pricing verified, security claim unresolved' gives the model material it can judge.

```
BAD
  research_status: 'probably enough'

BETTER
  sources_collected: 7
  official_sources: 3
  pricing_verified: true
  security_claim: 'unresolved'
```

C. STATE ACCESS CONTROL

A compact state packet should also enforce least privilege. The decision model receives only fields required for the contract. Secrets, personal information, proprietary code, and irrelevant account data should not be included merely because they exist in the parent agent context.

Field-level access rules make this boundary reviewable. They also allow the same underlying run state to produce different projections for routing, retrieval, verification, and approval contracts.

C. A COMPACT STATE PACKET

```
state = {
  goal: 'prepare a cited briefing on three tools',
  constraints: {
    publish: false, deadline: '09:00 UTC',
    max_sources: 12
  },
  completed_work: {
    sources_collected: 7, draft_exists: true,
    fact_check_complete: false
  },
  evidence: [
    {id:'s1', kind:'official_docs',
     supports:['pricing']},
    {id:'s2', kind:'launch_post',
     supports:['availability']}
  ],
  available_workers: [
    'research','write','fact_check','review'
  ],
  state_version: 'run_184:step_7'
}
```

The packet does not need to contain every event in the run. It needs the evidence required for the declared questions. Different decision families may receive different projections of the same underlying state.

D. FRESHNESS AND INVALIDATION

Every option set has an invalidation condition. Workers become unavailable. Buttons disappear. Budgets change. Files move. Permissions are revoked. Sources are updated. A decision over stale state is not merely a weak inference; it is an inference over an invalid graph.

Attach timestamps or versions to evidence that can become stale. Rebuild live options before evaluation. Prevent writes between snapshot creation and decision evaluation when consistency matters. If the system retries, require new evidence or a changed contract; repeating the same uncertain question against the same state is not learning.

E. STATE MINIMIZATION

Minimization is not aggressive deletion. It is a declaration of relevance. Code can pre-filter exact mismatches, retrieve likely evidence, and construct a small packet. Jev then resolves the remaining semantic ambiguity. This division prevents the model from rediscovering facts the program already knows exactly.

The packet should be inspectable by a human. If developers cannot explain why a field is present, the state has probably absorbed transcript history rather than decision evidence.

F. TESTING STATE PACKETS

State tests should remove, corrupt, and stale individual fields to measure which evidence actually drives the branch. If irrelevant history changes the answer, the packet is not sufficiently isolated. If removing a supposedly required field has no effect, either the field is unnecessary or the contract is not using it reliably.

Maintain fixtures for common state versions and edge cases. The fixture should include the expected route and a short human rationale, not a chain-of-thought trace. The aim is to verify the decision boundary and evidence sufficiency.

V.

CONFIDENCE, CONSEQUENCE, AND CALIBRATION

Confidence and Calibration

A. CONFIDENCE IS A ROUTING SIGNAL

Most systems flatten probability into a label too early. A result of 0.51 and a result of 0.99 may both become yes, even though they should not receive the same authority. The distribution should change what the harness does next.

A useful policy has three operating zones. High confidence plus low consequence may automate the declared branch. Medium confidence should improve the state by collecting evidence, running a deterministic check, narrowing options, or consulting a stronger model. Low confidence or high consequence should escalate.

CONSEQUENCE	CONFIDENCE	ROUTE
Low	>= 0.90	Automate the declared branch
Low or medium	0.70-0.89	Collect evidence or run a check
Any	< 0.70	Human review
Irreversible	Any	Human review

Table V. Thresholds are illustrative and must be calibrated per action class.

The same confidence should not control an internal queue and a public statement. Thresholds belong to the consequence class, not only to the output type. Irreversible actions may remain review-gated regardless of confidence.

B. IMPROVING THE STATE

A medium-confidence route should specify how the state will improve. Fetch another source, verify a file, run a permission check, ask the user a precise question, or reduce the option set. Confidence without a different next action is decoration.

Repeated evaluation with unchanged evidence can create false reassurance because several similar outputs look like consensus. The retry budget should purchase information, not merely another sample from the same uncertainty.

C. THRESHOLD GOVERNANCE

Thresholds are production configuration. Record their owner, rationale, evaluation window, consequence class, and rollback condition. A threshold chosen from a small demo set should never silently become permanent policy.

Monitor how much traffic falls near each threshold. A large mass just above an automation boundary makes the system sensitive to small calibration drift. In that case, improve the contract or widen the review zone before increasing autonomy.

C. CALIBRATION

Calibration asks whether events predicted near a probability occur at roughly that frequency on the target workload. A reliability diagram groups decisions into confidence bins and compares predicted confidence with observed correctness. Perfect calibration lies on the diagonal; a curve below it indicates overconfidence, and a curve above it indicates underconfidence. Automation depends on whether higher confidence actually identifies safer cases.

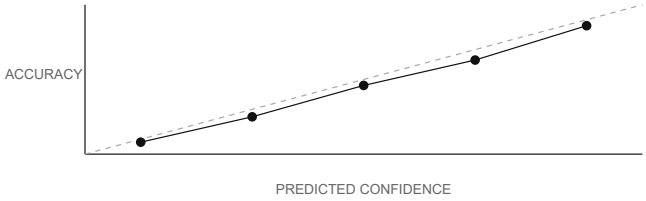

Fig. 2. Reliability is evaluated on labeled workload data, not inferred from confident language.

D. CALIBRATION METRICS

For binary Noul decisions, the Brier score measures squared error between probability and outcome. For Choice, top-label or classwise calibration can expose rare options that are systematically over- or under-confident. Segment metrics by contract version, consequence class, language, and rare option; a global average can hide the branch that matters most.

E. CONFIDENCE IS NOT AUTHORITY

Even a well-calibrated distribution is still a prediction. Policy, budgets, allowlists, user approvals, and deterministic checks remain downstream. Calibration makes automation measurable; it does not transfer authority to the model.

```
if consequence == 'irreversible':
    return HUMAN_REVIEW
if confidence >= auto_threshold:
    return selected_choice
if confidence >= evidence_threshold:
    return COLLECT_MORE_EVIDENCE
return HUMAN_REVIEW
```

F. CALIBRATION ROLLOUT

Calibration should be estimated in shadow mode, checked again after automation, and recalculated after material contract or state-schema changes. Confidence is conditional on the workload; moving the same contract to a new language, product area, or user population may invalidate the previous curve.

Use human labels selectively but consistently. Oversample rare, consequential, and near-threshold examples. Random samples alone may produce a reassuring aggregate while leaving the automation boundary poorly measured.

VI.

PARALLEL QUESTIONS AND STATE BOUNDARIES

Parallel Decisions

A. ONE SNAPSHOT, MANY QUESTIONS

Jev can evaluate multiple typed questions against the same state in one request. A router may ask for next worker by Choice, urgency by Score, and approval likelihood by Noul. Because the questions share one snapshot, the decision record is coherent and easier to inspect than a chain of independent generative calls.

```
response = system_one(
  state=state,
  questions={
    'next_worker': Choice(...),
    'urgency': Score(...),
    'requires_approval': Noul(...)
  }
)
```

Batching reduces repeated state transfer and prevents small timing differences from changing the evidence each question sees. It also exposes whether two contracts rely on incompatible projections of state.

B. INDEPENDENCE BOUNDARY

Parallel does not mean dependent. One question cannot consume another question's fresh answer inside the same evaluation. If the second decision requires a search result, tool output, or user clarification, perform that action, update the state version, and then ask again.

The boundary is simple: same evidence may be evaluated together; new evidence requires a new snapshot. Hiding a multi-step plan inside one label collapses the dependency and makes it impossible to verify which evidence supported the final branch.

C. SNAPSHOT DISCIPLINE

Every batch should carry a state version or content hash. The runtime should prevent relevant writes between snapshot creation and evaluation, or at least record the race. The decision receipt should identify which evidence each question was allowed to inspect.

Batching is not merely an optimization. It provides a semantic transaction boundary. All answers describe one version of reality, and any later action that changes reality produces a new boundary.

D. BATCHING ECONOMICS

A shared snapshot avoids sending the same state through several independent calls. More importantly, it makes disagreement visible: urgency, approval likelihood, and worker choice can be inspected together rather than reconstructed from separate traces.

Batch only questions that belong to the same latency and privacy class. A sensitive approval judgment should not inherit a provider route chosen for a low-cost internal classification merely because both inspect related state.

D. DECISION RECORD

```
{
  state_version: 'run_184:step_7',
  next_worker: 'fact_check',
  next_worker_distribution: {
    research: 0.05, write: 0.03,
    fact_check: 0.91, review: 0.01
  },
  urgency_score: 1.24,
  approval_probability: 0.82
}
```

The record retains distributions rather than only winners. A later audit can determine whether the chosen threshold was reasonable, whether a rare option was ignored, and whether confidence changed after new evidence arrived.

E. DEPENDENT SEQUENCE

```
STATE v7
  -> decide: evidence missing
  -> search: obtain official source
  -> STATE v8
  -> decide: evidence sufficient
  -> route: fact_check
  -> STATE v9
```

Each arrow that creates evidence must be visible in the trace. Otherwise the agent appears to reason continuously while the actual graph contains hidden state transitions and unrecorded assumptions.

F. CONCURRENCY AND SIDE EFFECTS

Independent read-only questions are natural candidates for parallel evaluation. Write-producing branches require stronger control. Two decisions may be semantically independent while their resulting actions contend for the same file, queue, budget, or external account.

The harness should separate decision parallelism from execution parallelism. It may evaluate routes together, then serialize or lock actions according to deterministic resource policy. A fast decision model does not remove ordinary distributed-systems concerns.

G. FAILURE MODES

Common failures include batching questions that actually depend on one another, mixing state versions in one record, evaluating after a write without rebuilding evidence, and logging only the final selected answer. Each failure weakens the audit trail even when the immediate branch appears correct.

H. TRACE RECONSTRUCTION

A complete trace lets an investigator replay the semantic transaction: load the state snapshot, recover the contract version, reproduce the declared options, inspect the distributions, apply the threshold configuration, and compare the resulting route with the action that executed.

If any link cannot be reconstructed, the decision is only partially observable. That gap should be treated as an operational defect even when no user-visible error occurred.

VII.

WHERE JEV BELONGS IN THE HARNESS

Harness Placement

A. BEFORE THE GENERATIVE MODEL

The first insertion point is task routing. A generative model is unnecessary when the system only needs to decide whether a request is simple, ambiguous, routine, high-stakes, or outside the available capability set. Jev can choose among the current workers or models, while code filters candidates that are unavailable, over budget, or forbidden by policy.

The router should receive the live model menu, current cost and latency budgets, task constraints, and consequence class. It should not contain a permanent prompt naming yesterday's model catalog. Routing over a stale menu is an invalid graph, not a weak inference.

B. BEFORE THE TOOL

The second insertion point is semantic risk classification. The LLM proposes a tool call. Jev evaluates the meaning of the proposed action against a declared rubric. Code then checks exact permissions, scope, budget, destination, and user approval before anything executes.

This separation prevents the classifier from becoming the policy engine. A low-risk prediction cannot override an allowlist, a repository boundary, an account permission, or a requirement for human confirmation.

```
proposal = llm.propose_tool_call()
judgment = jev.classify_semantic_risk(proposal)
route = policy.apply(judgment, proposal, user_scope)
if route == ALLOW:
    tool.execute(proposal)
elif route == REVIEW:
    queue_for_human(proposal)
else:
    block(proposal)
```

C. AFTER THE TOOL

The third insertion point is verification. A successful HTTP response or zero exit code proves that an operation completed, not that the user's goal was achieved. Verification should inspect new evidence: the artifact exists, the expected sections are present, claims have citations, the output path is correct, and no required approval remains unresolved.

D. TOOL SEMANTICS

A tool proposal should be normalized before judgment. Record the intended operation, target, destination, data class, reversibility, and external side effects. Semantic risk cannot be evaluated reliably from a command name alone when scripts, redirects, or nested calls change what the action actually does.

The normalized proposal also gives deterministic policy a stable interface. Jev interprets meaning; policy checks scope and authority against explicit fields.

D. BUILDER AND VERIFIER

The builder and verifier should share an artifact and a rubric, not a vague feeling of completion. The builder may use a generative model. The verifier may combine deterministic checks with Jev scores or choices. A failed check should route to repair, evidence collection, or escalation rather than an unbounded retry.

E. AROUND RETRIEVAL

Embeddings identify topical similarity but do not always identify evidence that is decision-relevant. A useful retrieval stack begins with cheap deterministic filters, creates an embedding shortlist, asks Jev to score relevance against the current question, constructs a small evidence packet, and finally calls the generative model.

This pipeline keeps exact facts in code and reserves semantic ambiguity for Jev. The trace can explain why each item survived. The generative model receives less noise, and the retrieval decision becomes testable independently from the generated artifact.

INSERTION POINT	JEV JUDGMENT	CODE AUTHORITY
Before model	Route task or model	Availability, budget, provider policy
Before tool	Estimate semantic risk	Permission, scope, approval
After tool	Pass, repair, or escalate	Artifact and invariant checks
Retrieval	Score decision relevance	Exact filters and access control

Table VI. Jev sits at semantic transitions; authority remains deterministic.

F. PLACEMENT DISCIPLINE

Do not place Jev at every edge simply because it is inexpensive. Begin with repeated semantic decisions whose errors are measurable and whose consequences are low. A decision node should remove a generative call, clarify a policy boundary, improve observability, or create a route that did not exist before.

If a node only restates an exact condition already known to code, it adds uncertainty without adding intelligence. If a node produces open-ended content, it is being used outside its intended role. The quality of the graph matters more than the number of Jev calls.

G. SECURITY-AWARE ROUTING

Model routing should consider trust in addition to quality, cost, and latency. The state packet can identify which data classes a branch may read. Code then removes providers that are not eligible for secrets, infrastructure, proprietary research, or personal data before Jev chooses among the remaining routes.

This preserves the system boundary: Jev may resolve semantic fit within the allowed set, but deterministic policy decides which set is allowed to exist.

VIII.

LIVE MENUS AND RETRIEVAL

Dynamic State

A. MENUS ARE RUNTIME STATE

The available actions in an agent change continuously. Browser controls appear and disappear after each click. Workers come online and go offline. Files are created or moved. Sources become stale. Budgets shrink. Permissions change. Queues fill. A decision model must choose from what exists now.

The option set should therefore be derived from live state immediately before evaluation. An internal menu written when the run began is not a reliable program representation. If a selected option no longer exists, the model did not necessarily infer badly; the harness evaluated an invalid graph.

```
controls = observe_current_page()
criteria = {
  control.id: control.semantic_description
  for control in controls
  if control.visible and control.enabled
}
criteria['stop'] = (
  'goal is complete or no safe action exists'
)
next_control = Choice(criteria=criteria)
```

B. LARGE OPTION SETS

Large menus should be reduced before semantic choice. Deterministic code removes impossible candidates. Retrieval or embeddings create a shortlist. Jev resolves the remaining ambiguity. This sequence is more stable than asking the decision model to scan hundreds of options that the program could have excluded exactly.

The shortlist process must preserve an escape hatch. Aggressive pre-filtering can remove the correct option before Jev sees it. Log both the original candidate count and the survivors so failures can be assigned to filtering, retrieval, or semantic choice.

C. DYNAMIC WORKER ROUTING

Worker menus should include availability, capability, cost, latency, trust policy, and current load. The semantic description explains when each worker is appropriate; deterministic fields rule out workers that cannot legally or operationally receive the task.

D. OPTION-SET TESTS

Test the menu builder separately from the semantic choice. Fixtures should cover unavailable workers, disabled controls, expired sources, exhausted budgets, and permission changes. The expected result may be a smaller menu or only the stop and review outcomes.

A correct Choice cannot recover an option removed by a faulty builder. Separating these tests prevents menu-construction defects from being blamed on Jev.

D. STALENESS HAZARDS

OBJECT	INVALIDATION	CORRECTION
Worker	Unavailable or overloaded	Refresh registry and load
Browser control	Hidden, disabled, or removed	Observe the current page
Source	Content or timestamp changed	Re-fetch and version
Budget	Spend increased	Read the current ledger
Permission	Scope revoked	Re-evaluate policy
File	Artifact moved or replaced	Resolve current path and hash

Table VII. Option validity is a property of current state, not the model.

E. RETRIEVAL AS DECISION SUPPORT

A retrieval item should survive because it contributes evidence to a declared question, not merely because it is topically close. The relevance contract may score whether a source directly supports a claim, whether it is authoritative enough for the consequence, and whether it is fresh enough for the decision.

The packet passed to the generative model should record source identifiers and support relationships. That structure allows the verifier to ask whether every material claim has evidence and whether the evidence type satisfies the contract.

F. OPTION DESCRIPTIONS

Descriptions must distinguish options rather than praise them. Two workers described as fast and capable create an under-specified menu. Criteria should state the evidence conditions under which each worker is the correct branch. Where options overlap, add a review or none outcome.

G. STOP AS AN ACTION

Dynamic menus should usually include stop. Without it, the agent is forced to keep acting after the goal is complete or when no safe action exists. Stop is not a failure; it is a declared outcome with criteria that can be evaluated and audited.

In browser and tool agents, a safe stop route is often more important than another recovery branch. It prevents the loop from spending tokens or creating side effects merely because an action menu cannot represent completion.

H. CACHING DYNAMIC STATE

Caching may be useful when option construction is expensive, but every cache entry needs an explicit invalidation rule. Time-based expiry alone is insufficient for permissions, budgets, and live controls. Prefer event-driven invalidation where the underlying system exposes a reliable change signal.

The decision receipt should identify the option-set version. When an incident involves a stale choice, the team can then distinguish inference error from invalidation failure.

IX.

EVALUATION AND PRODUCTION ROLLOUT

Evaluation and Rollout

A. BEGIN WITH ONE DECISION

Do not replace every hidden branch at once. Select one high-volume, low-consequence semantic decision whose correct answer can be labeled later. Internal ticket routing, worker selection, retrieval filtering, and completion checks are common starting points. Payment approval without human review is not.

B. DEFINE BEFORE INTEGRATING

Write the contract before calling the model. Specify state fields, instructions, options or rubric, fallback, confidence thresholds, consequence class, allowed action, escalation route, and version. If the team cannot agree on those fields, the branch is not ready for automation.

C. BUILD A REPRESENTATIVE SET

The evaluation set should contain normal examples, ambiguous cases, missing evidence, adversarial language, stale options, rare classes, and cases where no option fits. Include examples from the real distribution and deliberately difficult counterexamples. A polished demo set cannot reveal whether confidence is useful.

D. RUN IN SHADOW MODE

In shadow mode the existing production path remains authoritative. Jev evaluates the same decision but does not act. The system logs its distribution, selected answer, confidence, state reference, contract version, existing production answer, and later human label when available.

```
{
  decision_contract: 'ticket-router@3',
  state_hash: 'b476...',
  jev_model: 'jev-latest',
  answer: 'technical',
  confidence: 0.87,
  production_answer: 'technical',
  human_label: null,
  action_taken: false
}
```

Shadow mode reveals disagreements without exposing users to them. It also produces the data needed to test whether higher confidence actually corresponds to higher accuracy.

E. LABELS AND ADJUDICATION

Human labels need a written rubric and an adjudication path. If reviewers disagree systematically, the contract may be ambiguous rather than the model inaccurate. Preserve disagreement instead of forcing premature consensus; it indicates where the decision boundary requires clarification.

For consequential branches, have reviewers label both the semantic answer and the permitted route. This distinguishes a correct judgment from an unsafe automation policy.

E. EVALUATE THE WHOLE SYSTEM

Average decision accuracy is insufficient. Measure decision latency, decision cost, branch accuracy, downstream tool cost, recovery cost, human review rate, and completed-task rate. The business objective is cost and reliability per completed task.

METRIC	OPERATIONAL MEANING
Decision latency	Time to a typed result
Branch accuracy	Correct next action under the contract
Calibration	Observed correctness by confidence
Recovery cost	Cost after a wrong branch
Review rate	Load transferred to humans
Completed-task rate	End-to-end success

Table VIII. Per-call price is only one component of completed-task economics.

F. AUTOMATE THE SAFEST BRANCH

After shadow evaluation, automate one outcome with low consequence and strong calibration. Keep rare, uncertain, or consequential cases behind the existing review path. Compare production behavior with the shadow baseline and preserve an immediate rollback.

G. EXPAND ONE BOUNDARY AT A TIME

A typical sequence begins with internal routing, then retrieval filtering, completion verification, model selection, and low-risk tool gating. Consequential actions arrive later and remain surrounded by explicit policy and approval. This sequence keeps each new failure mode observable.

H. MONITOR DRIFT

Users, tools, options, and language change. Track accuracy and calibration by contract version and recent time window. Rising human-review rate may indicate degraded state quality, a stale menu, or a new class that the contract cannot express. Treat semantic drift like software drift: inspect, test, version, and roll back.

TypeSafe's published price can make individual decisions extremely cheap. That does not justify unnecessary calls. The economics are downstream. A small graph with high-quality boundaries is better than a dense graph of low-value classifiers.

I. ROLLOUT GUARDRAILS

Limit the first automated branch by traffic share, tenant, language, and consequence. Sample automated cases for review, retain the old path as a fallback, and define a rollback trigger before launch. Guardrails should be deterministic and independent of Jev confidence.

A rollout is complete only when the team can compare end-to-end task outcomes, not merely classifier agreement. The expected gain should appear in latency, cost, review load, or completed-task rate without an offsetting increase in recovery work.

X.

FAILURE MODES AND CORRECTIONS

Failure Modes

A. CONCLUSIONS STORED AS EVIDENCE

A state field says the research is probably enough. Jev then confirms the conclusion because it was presented as a fact. The correction is to store observable evidence: source count, source type, verified claims, unresolved claims, and freshness.

B. NO ESCAPE HATCH

Every listed option is wrong, but the contract requires one winner. The result looks confident because probability must sum over the closed menu. Add none, other, stop, or review whenever the menu can be incomplete.

C. ONE THRESHOLD FOR EVERY CONSEQUENCE

An internal classification and a public action use the same confidence threshold. This confuses prediction quality with authority. Define thresholds by action class, and keep irreversible actions review-gated.

D. STALE OPTIONS

The selected worker is unavailable, the browser control disappeared, or the source changed. Rebuild the option set from live state and attach a version to every decision.

E. BLIND RETRIES

The same uncertain decision is evaluated repeatedly with the same evidence. Nothing new is learned. Require each retry to add evidence, change the contract, narrow the menu, or escalate.

F. WINNER-ONLY LOGGING

The trace stores the chosen label but discards the distribution, threshold, state reference, and route. Calibration and audit become impossible. Store a complete decision receipt.

G. POLICY INSIDE THE CLASSIFIER

The question asks what should be allowed and application code blindly executes the answer. This gives probabilistic judgment authority. Keep permissions, budgets, scopes, and external side effects in deterministic policy.

H. EXACT RULES DELEGATED TO JEV

Attempt counts, dates, balances, allowlists, and exact strings are sent to a semantic model. Use code. Semantic judgment should not replace arithmetic or invariants the program already knows.

I. DISTINGUISH MODEL AND CONTRACT ERROR

When a decision is wrong, first ask whether the state contained the necessary evidence, whether the option set was valid, whether the instructions distinguished outcomes, and whether the threshold mapped to the correct route. Only after those checks should the defect be assigned to model capability.

This diagnostic order avoids compensating for harness errors with larger models, longer prompts, or repeated calls. Those responses increase cost while preserving the faulty boundary.

I. UNKNOWN VALUES HIDDEN INSIDE A LABEL

A contract asks Jev to produce a name, URL, identifier, or number not declared in the output schema. This is extraction or generation, not typed choice. Use the appropriate tool and then return to Jev when the set of outcomes is known.

J. A MULTI-STEP PLAN COMPRESSED INTO ONE ANSWER

A label implies search, verification, and execution without recording intermediate evidence. Decompose the graph. Run the tool that creates evidence, update state, and ask the next contract.

K. VERIFIER CHECKS TRANSPORT SUCCESS

The tool returned 200 or exit code zero, so the system declares success. Verify the goal instead: artifact presence, required structure, evidence, destination, and unresolved approvals.

L. CONTRACT CHANGES WITHOUT EVALUATION

A wording or option change is deployed as a harmless prompt edit. In reality it changes production logic. Version the contract, run historical examples, compare shadow behavior, and preserve rollback.

M. OPERATIONAL CORRECTION TABLE

OBSERVED SYMPTOM	INSPECT FIRST
High confidence, wrong branch	Evidence, menu completeness, calibration
Review rate rising	State freshness and new option classes
Costs rising	Wrong routes and recovery work
Repeated loops	Stop option and retry evidence
Unsafe action	Authority boundary and policy order
Schema valid, meaning wrong	Criteria and representative tests

Table IX. Many apparent model defects originate elsewhere in the harness.

N. SAFE OPERATING ORDER

The safe operating order is concise: Jev judges; code checks policy; the tool executes; the trace records. A production architecture should make each boundary visible in code and in the decision receipt.

The model provides a new primitive. The graph decides where it sits. The loop decides what happens after uncertainty. The harness decides what the system is allowed to do. The trace tells the team whether the architecture worked.

O. INCIDENT REVIEW

An incident review should reconstruct state, contract, distribution, threshold, policy, action, and outcome. It should identify the earliest incorrect boundary rather than the final visible failure. Corrective action may belong in state construction, menu generation, contract wording, calibration, policy, or execution.

Store representative incident states as permanent regression fixtures. A fix is incomplete until the revised system produces the intended judgment and route on those fixtures without weakening unrelated cases.

XI.

IMPLEMENTATION PLAYBOOK AND SOURCES

Implementation Playbook

A. TEN-STEP IMPLEMENTATION SEQUENCE

1. Map the agent loop and mark every hidden choice, score, yes-or-no judgment, retry, and stop decision.

2. Keep open-ended creation in the generative LLM and exact invariants in deterministic code.

3. Select one repeated, low-consequence semantic branch for the first Jev integration.

4. Define the decision contract: state, instructions, outcomes, escape hatch, thresholds, authority, escalation, and version.

5. Build compact evidence packets instead of passing transcript blobs.

6. Batch independent questions against one immutable state snapshot.

7. Route confidence into automate, improve-state, and human-review paths.

8. Run shadow mode and measure calibration on representative examples.

9. Automate the safest branch first and preserve rollback.

10. Expand one boundary at a time while monitoring completed-task economics.

B. LAUNCH CHECKLIST

[ ] The decision is semantic rather than generative or exact.

[ ] State is compact, current, and evidence-based.

[ ] Instructions define meaning explicitly.

[ ] Choice has an escape hatch where needed.

[ ] Score uses ordered verbal anchors.

[ ] Noul is interpreted as yes-or-no uncertainty.

[ ] Confidence changes the route.

[ ] Code retains authority over side effects.

C. CHECKLIST, CONTINUED

[ ] Independent questions share one state version.

[ ] Dependent questions follow a state update.

[ ] Thresholds are consequence-specific.

[ ] The full distribution is stored in the receipt.

[ ] The contract has run in shadow mode.

[ ] Representative tests include ambiguity and no-fit cases.

[ ] Every retry adds evidence or changes the graph.

[ ] The system can stop or escalate safely.

D. CONCLUSION

Jev is not a replacement for the generative model. It is a missing layer beside it. The LLM turns context into new work. Jev turns state into typed judgment. Code turns judgment into controlled action. Keeping those responsibilities separate reduces unnecessary generation, makes uncertainty operational, and gives the harness a traceable authority boundary.

The deepest improvement is not a single latency or cost number. It is the conversion of hidden judgments into versioned, testable components with evidence, thresholds, fallbacks, receipts, and rollback. That is the difference between adding another model call and engineering an agent.

SOURCES

[1] Rari. 'Jev Engineering: Stop Using LLMs for Every Decision.' X Article, 21 September 2026. https://x.com/0xwhrrari/status/2102020016539324501

[2] TypeSafe AI launch materials and posts by Diogo Almeida, as quoted and summarized in [1]. Reported latency, pricing, and 20-200x / 40-400x ranges are vendor claims and workload-dependent.

[3] 'Jev Engineering for Coding Agents.' Independent working note supplied with this project. The present handbook uses its harness framing while remaining aligned with the system-boundary, primitive, calibration, and rollout guidance in [1].

SUGGESTED AGENT PROMPT

Read this handbook and the companion article. Map every hidden choice, score, yes-or-no judgment, and side-effect boundary in this repository. Propose one low-consequence decision contract to run in shadow mode first.