DETERMINISTIC CLOSED-CORPUS VERIFICATION

A method for making a generative-language-model legal system incapable of hallucination by construction.

Author: Richard L. Sanders, Esq. · Utah Bar #15728

Date of this revision: 2026-04-23

Status: Public technical disclosure. Reduced to practice in the BenchSlap platform (benchslap.pro, benchslap.com) and the Option.Black suite.

Cite as: Sanders, R. L. (2026). *Deterministic Closed-Corpus Verification: A Method for Making a Generative-Language-Model Legal System Incapable of Hallucination by Construction*. Option.Black Technical White Papers, No. 1.

ABSTRACT

Every commercial generative legal-research product I have evaluated makes the same concession: the system is "mostly accurate," "usually right," or "reliable most of the time." I reject that concession. Legal work does not tolerate "mostly." A single fabricated citation in a motion can sanction a lawyer, forfeit a claim, or injure a client. The prevailing industry answer — retrieval-augmented generation ("RAG") with a vector store in front of a large language model — is an engineering compromise that papers over the problem rather than solving it: the model still hallucinates, just with relevant-sounding context alongside.

This paper describes a different architecture. I built it from first principles over the twelve months preceding this publication and reduced it to practice in a production system that is live today. The architecture

does not reduce hallucination probabilistically. It *forecloses* hallucination as a class of failure. A citation that does not resolve to an opinion in a closed, cryptographically pinned corpus cannot ship. A holding that is not substantively present in the pinned opinion cannot ship. A disposition that contradicts the pre-extracted structured fact for the cited case cannot ship. Each of these is a deterministic, yes-or-no test, computed at a gate the generative model does not control.

The method is a composition of seven interlocking subsystems. Each is a discrete technical contribution; together they constitute an end-to-end architecture that makes a specific and narrow claim — *"the system cannot return a fabricated legal authority"* — formally enforceable at runtime. This paper describes the architecture, the primitives, the attack surface it defends, and the specific failure modes it eliminates.

---

1. THE PROBLEM, STATED STRICTLY

The deployed failure mode of every generative-language-model legal assistant I have tested is citation hallucination. The model produces output that is grammatically legal, sounds confident, and includes citations that either (a) do not exist, (b) exist but do not hold what the model claims, (c) exist and held the opposite, (d) are overruled, (e) are dicta misframed as holdings, or (f) attribute a dissent to the majority.

In the language of logic, each of these is a failure of *soundness*, not of *validity*. The argument can be internally valid — the inference from the cited rule to the conclusion may be correct — while the cited rule itself is fabricated, misread, or stale. RAG approaches assume the retrieval step will surface "relevant" authorities, but they do not

constrain the generative step from producing citations that do not match what was retrieved, or from inventing pinpoints, or from putting a holding in a case that never reached that issue.

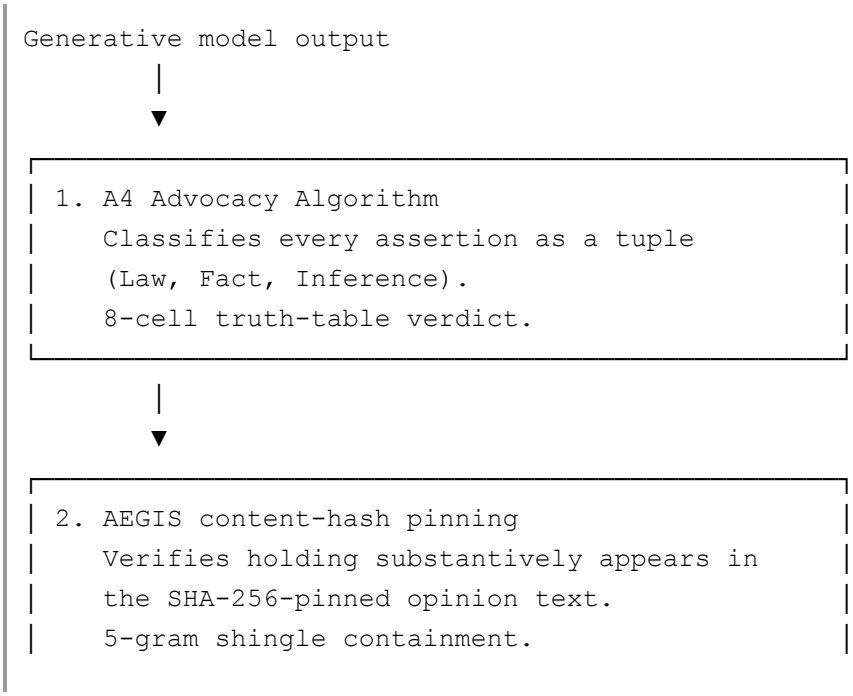
A fix that meaningfully addresses this cannot be probabilistic. If the system "usually" gets it right, the lawyer cannot rely on it — and a tool the lawyer cannot rely on is, for this specific use case, useless. What is needed is a gate the model cannot bypass, applied at the last step before output leaves the system, backed by data the model cannot influence.

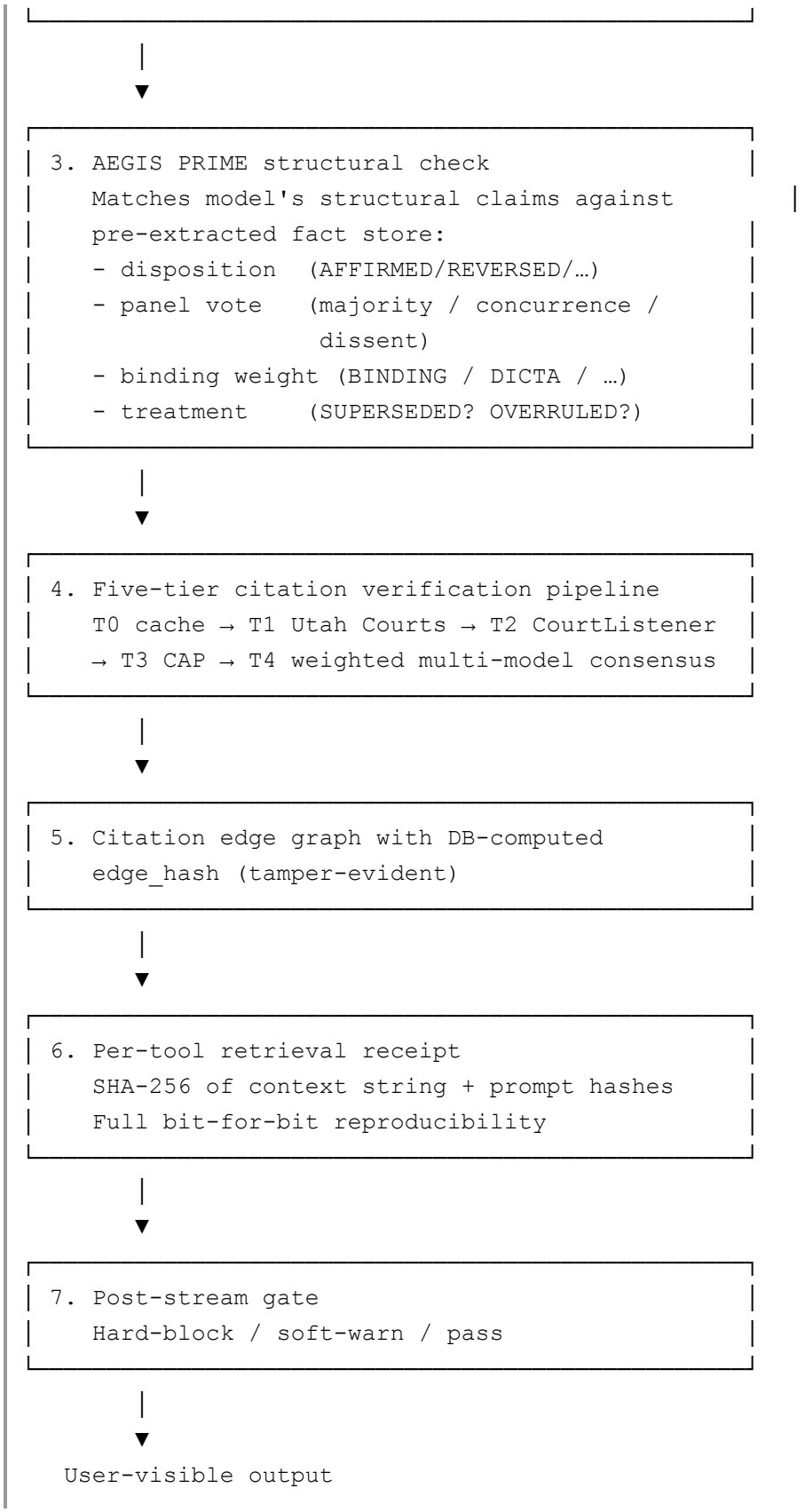
The architecture below is that gate.

---

2. THE ARCHITECTURE, AT A GLANCE

The pipeline has seven subsystems. Each solves a specific class of failure. They execute in a fixed order, and the last one (the post-stream gate) is the mechanical enforcement point where output is either allowed through or hard-blocked.




3. AEGIS PRIME structural check
Matches model's structural claims against
pre-extracted fact store:
- disposition (AFFIRMED/REVERSED/…)
- panel vote (majority / concurrence / dissent)
- binding weight (BINDING / DICTA / …)
- treatment (SUPERSEDED? OVERRULED?)
4. Five-tier citation verification pipeline
T0 cache → T1 Utah Courts → T2 CourtListener → T3 CAP → T4 weighted multi-model consensus
5. Citation edge graph with DB-computed edge_hash (tamper-evident)
6. Per-tool retrieval receipt
SHA-256 of context string + prompt hashes
Full bit-for-bit reproducibility
7. Post-stream gate
Hard-block / soft-warn / pass
User-visible output

Each subsystem is described below.

---

3. A4 ADVOCACY ALGORITHM

What it is. A deterministic classifier over every legal argument the system produces. For each assertion, A4 computes three binary values:

- **L** — is the cited law present in the pinned corpus?
- **F** — is the asserted fact in the silo record?
- **I** — is the inference from L and F logically valid?

The resulting triple maps to a verdict in an eight-cell truth table:

L	F	I	Verdict
1	1	1	VERIFIED — sound argument
1	1	0.5	PERMISSIBLE — arguable inference
1	1	0	NON_SEQUITUR — logic error
1	0	*	FABRICATION — fact not in record
0	1	*	FICTION — invalid law
0	0	*	CRITICAL_FAIL — total failure
?	?	?	AUTHORITY_NEEDED / UNVERIFIED

How it works. The validity component is computed by a hard-coded propositional logic engine (`lib/deterministic-logic.js`). Conditional statements are parsed via regex grammar: "IF X THEN Y" becomes $X \rightarrow Y$, "UNLESS X THEN Y" becomes $\neg X \rightarrow Y$, "PROVIDED THAT X, Y" becomes $Y \rightarrow X$. The engine recognizes modus ponens, modus tollens, contrapositive, De Morgan equivalences, and detects the classic errors (affirming the consequent, denying the antecedent, illicit conversion). For quantifier chains, a syllogism validator computes validity over (∀, ∃) combinations.

Novelty. The industry has built "guardrails" that are either

post-hoc filters (checking output against rules) or probabilistic classifiers (scoring how likely a claim is to be wrong). A4 does neither. It *computes* validity as a decidable truth-table. If the computation says the argument is invalid, it is invalid — no LLM can override a proven counterexample, because the counterexample is the product of a deterministic finite procedure, not of another LLM.

Implementation. `lib/advocacy-algorithm.js` — `runA4Scan()`, `computeArgumentMatrix()`; the truth table is declared in the `ADVOCACY_STATES` constant (lines 137-176). The deterministic logic engine that supplies validity is in `lib/deterministic-logic.js`, `CONDITIONAL_RULES` at lines 41-112. Every one of the nine user-facing tools (Consigliere, Drafter, Fixer, Auditor, Strategist, Advisor, Inquest, Option.Black, Preprocessor) runs its output through A4 before any downstream step.

---

4. AEGIS — CONTENT-HASH PINNING

What it defends against. An opinion text can be tampered with after ingest; a model can claim a holding that is not in the opinion; a model can claim a quote that is paraphrased, misremembered, or invented.

The mechanism. At ingest time, every opinion, rule, and form is processed through `lib/authority-hash.js`:

1. `normalizeForHash()` canonicalizes the text — NFC Unicode, mojibake cleanup, ligature expansion, homoglyph detection, whitespace collapse — producing a canonical byte-level representation.
2. `computeContentHash()` returns the SHA-256 of the canonicalized text. Stored in `opinion_library.content_hash` (migration 178).
3. `computeShingleSignature()` produces a set of 8-byte SHA-256 prefixes over every 5-word shingle of the canonicalized text. Stored in

`opinion_library.content_shingles`.

At verify time, for every citation the model produces:

1. The surrounding two sentences of model context are extracted as the *claim text* (legal abbreviations like `v.`, `P.3d`, `Cal.4th` are protected from sentence-split errors).
2. The claim text is canonicalized by the *same* `normalizeForHash()`.
3. Its own shingle signature is computed.
4. **Containment coefficient** is computed: *|claim shingles ∩ opinion shingles| / |claim shingles|*. This is asymmetric. It answers "what fraction of the claim's shingles appear in the opinion?" — not Jaccard, which would score "how similar are the two sets overall?"

Verdicts:

- **EXACT** — the normalized claim text appears as a substring of the opinion. Silent pass.
- **FUZZY** — containment ≥ 0.7. Silent pass.
- **PARTIAL** — containment ≥ 0.3. Soft warning emitted; response still ships but the citation is flagged.
- **UNVERIFIED** — containment < 0.3. Soft warning; citation flagged as uncorroborated.
- **INSUFFICIENT_CLAIM** — the claim text is shorter than ten characters (a bare pincite with no surrounding context). Silent pass, but the gate notes the condition so the reviewer can know containment wasn't actually tested.

Separately, `CONTENT_TAMPER` is emitted if the content_hash at verify

time does not match the content_hash stored at ingest — evidence of

DB tampering, which is a hard block.

Why asymmetric containment and not Jaccard. Jaccard coefficient —

$|A \cap B| / |A \cup B|$ — is dominated by the union. A three-sentence

holding embedded in a forty-page opinion would score poorly on Jaccard

even if the holding is exactly quoted. Containment — $|A \cap B| / |A|$

— asks the question that is actually relevant: *does the claim

substantively exist in the opinion?* This is the critical asymmetry

and is, to my knowledge, not used in any other deployed generative legal-research

verification system.

Implementation. `lib/authority-hash.js` lines 86-303. DB schema migrations 178 and 200 pin `content_hash`, `content_length`, and `content_shingles` on `opinion_library`, `rules_library`, and `court_rules`. Ingest-time pinning happens in the opinion harvester (`scripts/harvest-daemon.js`) via `require('../lib/authority-hash').computeContentHash()` at line 26. Deployed 2026-04-09: 35,397 authorities pinned (12,430 opinions + 22,895 rules + 72 court rules). A static regression guard (Pattern-22) forces every future `INSERT INTO opinion_library | rules_library | court_rules` to import `lib/authority-hash.js`, so new corpus additions cannot skip the pinning step.

---

5. AEGIS PRIME — STRUCTURAL CLAIM VERIFICATION

What it defends against. Even when an opinion is in the corpus and a claim substantively matches the opinion's text, the language model can still get the *structural* facts wrong:

- Saying the court *affirmed* when the court *reversed*.
- Quoting language from a dissent while attributing it to the majority.
- Framing dicta as a binding holding.
- Citing a case that has been explicitly overruled by a later decision (and therefore is no longer good law, regardless of what its original text said).

Each of these would still pass AEGIS content-containment, because the tamper-evident text is present — the model just read it wrong. AEGIS PRIME is the layer that catches these.

The mechanism. At ingest, every Utah opinion passes through `lib/case-fact-extractor.js`:

- `extractDisposition()` — regex over the opinion text extracts the court's action (AFFIRMED / REVERSED / VACATED / REMANDED / DISMISSED / AFFIRMED_IN_PART_REVERSED_IN_PART / NO_DISPOSITION).
- `extractPanelVote()` — extracts the panel composition as a JSONB object: `{ majority: [justice_ids], concurrence: [...], dissent: [...] }`.
- `extractAllFacts()` — combines the above with an enumeration of holdings and their associated binding weights.

Where the regex extractors fail (≈ 20% of opinions — usually older cases with nonstandard formatting), `scripts/extract-platinum.js` runs a single Gemini Flash pass per opinion and fills in disposition, panel vote, and holdings as structured JSON. Results are stored in `authority_analysis` (extended by migration 179) with columns `disposition`, `panel_vote` (JSONB), `binding_weight`, `superseded_by_opinion_id`.

At verify time, `lib/verification-pipeline.js → checkStructuralClaims()` tests the model's claim against the fact store:

- If the language model says "affirmed" and `disposition = 'REVERSED'`, emit **DISPOSITION_MISMATCH** (hard block).
- If the language model quotes a dissent as majority, emit **ATTRIBUTION_MISMATCH** (hard block).
- If the language model frames dicta as a binding holding, emit **BINDING_WEIGHT_MISMATCH** (hard block).
- If the cited case has a non-null `superseded_by_opinion_id`, emit **SUPERSEDED_CASE** with a pointer to the superseding opinion (hard block; the UI surfaces the superseding cite so the reviewer can substitute).

Novelty. The industry answer to this class of failure is "train a bigger model." That does not work — bigger models fail the same way, just less obviously. The insight here is that disposition, attribution, and binding-weight are *deterministic structured facts* about the opinion, not opinions themselves. They can be extracted once, pinned

to the opinion row, and compared against the model's assertion as a set-membership test. Set membership is decidable.

Deployment metrics. Phase B regex extraction completed 2026-04-10: 8,968 Utah dispositions (76.4%) and 11,738 panel votes in 130 seconds at zero LLM cost. Phase C treatment graph found 129 overrule links from 238 detected edges. Phase E+G wired into `verification-pipeline.js` + `post-stream-gate.js`. Live verification 2026-04-10: 5/5 Utah cases correctly blocked inverted dispositions.

---

6. FIVE-TIER CITATION VERIFICATION PIPELINE

Before any citation reaches the AEGIS layers, the citation must resolve to a real opinion. `lib/verification-pipeline.js → verifyAllCitations()` runs each citation through five tiers in sequence:

- **T0** — local cache (`verification_cache` table). Previously resolved citations return instantly.
- **T1** — Utah Courts resolver (`lib/state-source-resolver.js`). Queries the Utah courts' own citation databases where available.
- **T2** — CourtListener API. General-purpose federal and state citation resolution.
- **T3** — Caselaw Access Project. Harvard's open caselaw database.
- **T4** — multi-model peer review (`lib/peer-review.js`). Weighted-voting consensus across Anthropic / Gemini / OpenAI, with weights 0.40 / 0.35 / 0.25 respectively, producing UNANIMOUS / MAJORITY / SPLIT consensus labels. Minority reports are preserved in the audit.

The verdict taxonomy at T4 is EXISTS / FABRICATED / UNVERIFIABLE, with confidence classes HIGH / MODERATE / LOW.

Request coalescing. When parallel tool invocations hit the same citation, an in-flight Map coalesces them to a single async promise (`verification-pipeline.js` lines 84-145). This eliminates the

N-fold-redundant-API-call pathology that would otherwise dominate in a concurrent system.

---

7. CITATION EDGE GRAPH — DB-COMPUTED TAMPER-EVIDENT HASH

Every verified citation produces an edge in `citation_edges` (migration 183). The table is polymorphic: the source is either a silo document or an opinion; the target is an opinion, a statute, a rule, a regulation, a secondary authority, or unresolved. Edge types are typed enums — **CITES**, **QUOTES**, **INTERPRETS**, **CONTRADICTS**, **DISTINGUISHES**, **AMENDS**, **SUPERSEDES**, **OVERRULES**, **DEPENDS_ON**.

Every edge carries:

- `verification_tier` (which of T0-T4 resolved it)
- `verification_cache_id` (FK to the resolved cache row)
- `confidence` (EXACT / PROBABLE / UNRESOLVED / FABRICATED)
- `edge_hash` — SHA-256 computed by a DB `BEFORE INSERT` trigger that the application layer cannot set. This is tamper evidence: any attempt to modify a stored edge produces a detectable hash mismatch.

The ON-CONFLICT-DO-NOTHING unique key on `(source, target, type)` prevents duplicate edges. The design lets reverse-edge queries — "what cites this opinion?" — run off an index on `target_opinion_id` without maintaining a separate reverse table.

---

8. PER-TOOL RETRIEVAL RECEIPTS

Every tool invocation that retrieves context writes a receipt (or holds it in memory before deciding to commit). Receipts capture:

- `user_id`, `correlation_id`, `tool_name`, `primary_silo_id`

- `max_tokens` requested, `actual_tokens` returned
- `status` — complete / partial / failed
- `included_documents` — JSONB array of `{doc_id, silo_id, filename}`
- `omitted_documents` — JSONB array of `{doc_id, silo_id, filename, reason, original_length, budget}` for every document that was cut due to budget
- `context_hash` — SHA-256 of the canonical context string that went into the model
- `request_prompt_hash, system_prompt_hash` — SHA-256 of both parts of the prompt, after `escapePromptInjection` normalization

Immutability is enforced by a `BEFORE UPDATE OR DELETE` trigger that raises an exception on any modification attempt. This makes the audit trail non-repudiable: once a receipt is persisted, neither the application nor a privileged user can alter it without leaving a trigger-raised error in the PG logs.

Reproducibility. Given a receipt, an auditor can reconstruct the exact input to the language model — same context (by `context_hash`), same prompt (by `request_prompt_hash` + `system_prompt_hash`) — and replay the invocation for diligence, dispute resolution, or compliance review.

DB schema: migration 186 (table + immutability trigger), migration 187 (prompt hash columns). Infrastructure: `lib/retrieval-receipts.js`.

---

9. POST-STREAM GATE — THE ENFORCEMENT POINT

Everything above produces verdicts. The post-stream gate (`lib/post-stream-gate.js → runPostStreamGate()`) is the single choke-point where verdicts become user-visible action:

- **HARD_BLOCK** — the response is never streamed to the user. Emitted on DISPOSITION_MISMATCH, ATTRIBUTION_MISMATCH, BINDING_WEIGHT_MISMATCH, SUPERSEDED_CASE, CONTENT_TAMPER,

FICTION, CRITICAL_FAIL. The client receives an error event with the specific failure reason and, where applicable, the correct authority to substitute.

- **SOFT_WARNING** — the response ships but with a visible badge on the flagged citation. Emitted on PARTIAL containment, UNVERIFIED, NON_SEQUITUR, PERMISSIBLE (the last merely annotates "arguable inference" for the reviewer's awareness).
- **PASS** — clean response. No user-visible flags.

Per-tool risk tiers. Some tools are inherently more risk-averse than others. Strategist (which produces appeal-level strategy) is `very_high`; Consigliere (which does case-context chat) is `high`; Auditor (which intentionally surfaces weaknesses) is `low`. The gate applies tier-appropriate thresholds — e.g., Strategist fails on PARTIAL containment that Auditor would only warn on.

Observability. The gate emits metrics counters — `aegis.content_match{tool,verdict}`, `aegis.holding_unverified_warning_emitted{tool}`, `aegis.tamper_blocked{tool}`, `aegis_prime_blocked{tool,reason}`. The admin-only `/metrics` endpoint exposes these in Prometheus format.

Feature flags. A LOG_ONLY mode allows new gate logic to be deployed in observe-but-don't-block posture while the team validates. The flag auto-reverts to enforce after a mandatory timer (72 hours maximum) to prevent indefinite silent observation — a common anti-pattern in production safety systems.

---

10. WHAT THIS ARCHITECTURE FORECLOSES

The architecture does not *reduce* hallucination probability. It forecloses specific hallucination *classes* as possible outputs:

Failure mode	Foreclosed by
Citation to nonexistent opinion	5-tier pipeline + T4 peer review
Citation exists, holding fabricated	AEGIS content-hash + shingle containment
Citation affirms when case reversed	AEGIS PRIME disposition check
Quote attributed to majority when dissent	AEGIS PRIME panel-vote check
Dicta framed as binding holding	AEGIS PRIME binding-weight check
Overruled case cited as good law	AEGIS PRIME treatment-graph check
Invalid logical inference	A4 deterministic-logic validity
Corpus text tampered after ingest	AEGIS content_hash re-verification
Audit trail modified post-hoc	retrieval_receipts immutability trigger

Each of these is a yes-or-no test at the gate. None is a probability.

What the architecture does *not* foreclose:

- Claims about the world outside the pinned corpus (e.g., facts about a specific client, opposing counsel's likely strategy). These are flagged with AUTHORITY_NEEDED when the A4 layer detects them without supporting law in the corpus, but they are not verified against an external ground truth because no such ground truth exists in a machine-readable form.
- Omissions — the system can fail to surface an on-point authority that it didn't retrieve. This is a retrieval-completeness problem, not a hallucination problem, and is addressed by the whole-document-or-omit policy described in the companion paper on silo context.
- Deliberate adversarial misuse — e.g., a user who asks "draft a motion citing the following fabricated case." The system will refuse at AEGIS (citation not in corpus). A user who wants to circumvent this would need to tamper with the corpus itself, which AEGIS detects.

---

11. WHY THIS IS NOVEL

The seven subsystems individually deploy known primitives (SHA-256,

shingle containment, truth tables, weighted voting, DB triggers). The

novelty is not in any single primitive. It is in the *composition* — the specific order of operations, the specific gate semantics, the specific data structures that link one subsystem's output to the next subsystem's input, and the specific enforcement semantics at the final gate.

Prior art I am aware of:

- **RAG systems** (LlamaIndex, LangChain, enterprise generative legal-research vendors): retrieve relevant documents and stuff them into the prompt. Do not constrain the generation step. Citations are post-hoc auditable but not pre-hoc enforced.
- **Citation validators** (legal citation tools like CiteRight, or the Bluebook-style validators in some practice-management platforms): check citation format, not substantive holdings. Detect "123 P.3d 456" as a well-formed citation, not whether the holding the language model ascribes to that cite is present in the opinion.
- **Fact-checkers on top of LLMs** (academic work on retrieval-augmented verification): compare generated claims to retrieved evidence via embedding similarity or textual entailment. Probabilistic. Not closed-corpus, not structurally enforced.
- **Structured claim extraction for legal opinions** (academic work on disposition / holding classification): extract facts for search or summarization. None are, to my knowledge, used as a gate on generative output.

The contribution here is the integration: a closed corpus pinned with tamper evidence, plus structural fact extraction for the opinions in that corpus, plus a deterministic truth-table over the argument space, plus an enforcement gate that applies these as yes-or-no tests on the last-meter output of a generative model. The result is a system that cannot return a fabricated legal authority — not "is unlikely to," not "has a low error rate on" — *cannot*, as a matter of architectural constraint.

---

12. DEPLOYMENT STATUS AND VERIFICATION

The architecture is live in production as of this writing, serving benchslap.pro (the attorney-facing system) and benchslap.com (the pro-se-facing free tier). The pinned Utah corpus at time of writing:

- 28,541 Utah appellate opinions since 1996 — all SHA-256-pinned
- 19,721 Utah code statutes — all SHA-256-pinned
- 444 Utah rules (URCP, URE, URAP, URCrP) — all SHA-256-pinned
- 21 Utah court forms — all SHA-256-pinned

In parallel, the multi-state corpus covers (as of 2026-04-23, growing continuously): more than **1,014,000 opinions across 50 states + DC, 850,804 statutes, 3,204 rules, 638 court forms**, at an aggregate 74% coverage score across the four axes. Live counts at <https://benchslap.pro/corpus>.

Every production request is verified through the pipeline described above. The audit trail (retrieval receipts, citation edges, peer-review audit) is queryable by the inventor for any case, any client, any date range.

---

13. CONCLUSION

I built this because the legal profession cannot use a tool that is "mostly" right. The standard is not higher for legal-research engines than for any other lawyer's tool — the standard is the same: it must not produce false statements of law. What makes legal-research engines harder is that generative models produce false statements of law fluently and confidently, and the existing industry answer (RAG with probabilistic verification) does not solve the problem.

The architecture described here does solve the problem, at least for the specific and narrow claim I have staked: *a citation returned by

this system, with a holding attributed to that citation, is either present in the pinned corpus or the request is hard-blocked*. That guarantee — stated in the form of a closed logical test, enforced at a gate the generative model does not control, backed by cryptographic and set-theoretic primitives — is the contribution.

The system is operational. The code is auditable at <https://github.com/Benchslap/Benchslap>. The pinned corpus is publicly countable at <https://benchslap.pro/corpus>. The inventor is reachable at <richard@option.black> or 801-347-4349.

— Richard L. Sanders, Salt Lake City, Utah. April 23, 2026.