Evidence-Grounded Verified Agentic Reasoning: A Path Toward Eliminating LLM Hallucination in Empirical Inference via Tool-Attested Kernel Proofs

Junyu Ren [1]

Abstract

Tool access alone does not make LLM empirical reasoning governable: accepted outputs need not descend from attested evidence, and accepted deductions need not hold up under formal scrutiny. We present EG-VAR (Evidence-Grounded Verified Agentic Reasoning), a Lean 4-based tool-calling architecture in which the Lean kernel is the sole minter of VERIFIED claims via tool-attestation axioms and declared source lifts. Every verified output structurally descends from an attested tool call (Theorem 3.1) and a kernel-checked chain of valid inference (Theorem 3.2); residual outputs are honest ABSTAIN with a replayable audit trail. On a subcollection of TableBench numerical reasoning (n=120), EG-VAR attains 120/120 versus a 95% same-tool baseline; on counterfactual stress tests (5 domains $\times$ 2 models), EG-VAR stays 100% source-faithful while same-tool drops to 80–90% (no-tool 50–80%). With the LLM as deployment-time formalizer, residual semantic-formalization error is 3.3% on Sonnet and 1.7% on Opus. We position EG-VAR as a technical-governance interface for high-stakes empirical claims: a formal sidecar makes the target proposition, source scope, evidence boundary, proof obligation, and abstention condition auditable, eliminating unsupported VERIFIED outputs today while turning formalization errors, lift and source-authority disputes, ambiguities, and abstentions into explicit audit targets. Over time, typed sidecars in datasets, APIs, public records, and AI-generated documents can amortize this formalization burden into reusable infrastructure.

[1]Committee on Computational and Applied Mathematics, University of Chicago, Chicago, IL, USA. Correspondence to: Junyu Ren <junyuren@uchicago.edu>.

Accepted to the TAIGR Workshop at ICML 2026. Copyright 2026 by the author.

1. Introduction

LLMs hallucinate on empirical facts when making inference, fabricating or reproducing outdated ones (Huang et al., 2025). Tools and external memories were introduced as a mitigation, letting the LLM consult an authoritative source rather than rely on its parametric prior. But tool access alone does not yield verifiable empirical claims: frontier models exhibit strong confirmation bias, dismissing retrieved evidence that conflicts with parametric memory (Xie et al., 2024), and even state-of-the-art models struggle to stay faithful to counterfactual context (Ming et al., 2025); our own counterfactual stress tests on tables reproduce the pattern at $\sim$12–15%, robust to an explicit "according to the table" source-anchoring cue (Section 4.2).

Tool access leaves two structural failures uncovered: accepted outputs need not descend from attested evidence, and accepted deductions need not hold up under formal scrutiny. EG-VAR targets each with a structural contract: every VERIFIED output must descend from an attested tool call (Theorem 3.1), and every accepted proof step must type-check in the Lean kernel under the declared axiom set (Theorem 3.2). VERIFIED means attested evidence plus kernel-checked derivation; everything else surfaces as honest ABSTAIN with a replayable audit trail. We call this combination a *verified empirical claim*: a claim any auditor can later check without re-running the model.

Concretely, given a claim like "Brazil scored on average x more goals in the 2018 opener than Italy" against a spreadsheet, EG-VAR compiles the claim with indeterminate answer into a typed goal in Lean 4. An LLM uses tool calls to fetch attested values and compute statistics from the table, reasons and infers an answer value derived from the attested evidence, and proposes a Lean tactic encoding this derivation as a proof; the Lean kernel type-checks the tactic. Accepted: publish the answer with a machine-checkable proof; rejected: abstain. A third party re-checks offline by re-running Lean; the LLM need not be re-run.

Our contribution. We present **EG-VAR** (Evidence-Grounded Verified Agentic Reasoning), a source-general

architecture for empirical claims grounded in attested external data, with tables as the first end-to-end instantiation. To our knowledge this is the first proof-assistant-verified architecture in Lean 4 for this setting. The LLM is one untrusted layer in a four-layer stack: a deterministic *tool layer* (L1, attested storage queries), a per-source *formalization layer* (L2, audited semantic lifts from storage to world facts), a *Lean 4 kernel* that mints *Verified* world claims solely via attestation-derived rules, and the *solver LLM* that chooses the answer witness, substitutes it into the claim's dependent-sum goal, and proposes a proof the kernel checks. The key architectural commitment is that `mkVerified` (the sole constructor for `Evidence Verified` w) requires an `Attested_T` from the runtime, so every verified output structurally descends from a tool call. The certified answer is extracted as the witness of a dependent sum (Martin-Löf, 1984) in the Curry–Howard tradition (Howard, 1980): a question "which entity has property P?" becomes a goal of type $\Sigma\,(e\ :\ \mathrm{String})\,(v\ :\ \mathrm{Value}), \mathrm{Evidence\ Verified}\,(\ldots\, e\, v)$, and the surface answer is the first projection of the kernel-checked witness. Verified outputs come with replayable proof traces; unverified claims abstain rather than guess. Unlike symbolic factual verifiers, execution-attestation systems, and math theorem provers (Section 2), EG-VAR combines proof-assistant checking with attested empirical grounding.

Why a proof assistant for empirical grounding? The standard objection is that proof assistants are overkill for empirical reasoning over external sources. We disagree: existing LLM-driven theorem-proving systems (AlphaProof (Google DeepMind, 2024), Aristotle (Achim et al., 2025), DeepSeek-Prover (Xin et al., 2025), Hilbert (Varambally et al., 2025)) already show that the substrate is not the bottleneck; the bottleneck is audited per-source semantic lifting, which our formalization protocol addresses (Section 3.5; instantiated here on tables). This cost is amortized: each per-source lift is audited once and reused across all subsequent claims grounded in that source, so formalization never enters the per-query inference loop.

Scope of the empirical work. All empirical results in this paper use the tabular instantiation and evaluate on TableBench (Wu et al., 2025). This is the first end-to-end demonstration of the L1/L2/lift pattern on a single substrate, not the conceptual boundary of the framework; cross-substrate validation (SQL views, typed APIs, knowledge graphs) is future work.

Empirical findings. On TableBench unambiguous claims (Tier 1 gold-goal benchmark, n=120), EG-VAR attains 120/120 (100%); the same-tool same-model baseline plateaus at 114/120 (95.0%) and the table-only single-shot baseline at 107/120 (89.2%). On a 5-domain counterfactual stress panel (Tier 1.5 binary-compare and direction claims, n=20 per panel, two models, two injection regimes), EG-VAR remains 100% source-faithful in every cell; on Sonnet the same-tool baseline drops to 80–90% and the no-tool baseline to 50–80%, depending on injection magnitude. Under full end-to-end deployment with the LLM itself as the formalizer (Tier 2, n=120), the pure formalizer-error rate is 3.3% on Sonnet and 1.7% on Opus, with the remaining disagreements split into logged ambiguities (4.2%) and surfaced benchmark-artifact issues (1.7%) that are auditable rather than silent (Section 4.3). The discriminator is robust to model strength, source cue, and flip magnitude.

A path toward eliminating LLM hallucination. The title claim has a precise structural meaning (Theorem 3.1): no VERIFIED output can appear without an attested tool leaf, under any formalizer. Semantic mis-formalization remains non-zero at current LLM capability (3.3% on Sonnet, 1.7% on Opus), but this accuracy axis is addressable by fine-tuning on the curator-grade (claim, gold goal) pairs the paper releases (code supplement in Section G; curator pipeline summary in Section 3.6 with full proposer/critic/kernel enumerate in Section G.2; case studies in Section I.2) without weakening the structural guarantee.

Formalization flywheel. The long-run bet is not that users will manually write Lean. It is that empirical communication can gradually acquire *formal sidecars*. In the near term, EG-VAR's curator corpus supplies supervised and verifier-guided training data for better NL-to-formal translation. In the medium term, many sources already move along a substrate spectrum from CSV to SQL views, OpenAPI schemas, knowledge graphs, and typed registries, reducing the per-source lift burden (Section K.3). In the long term, LLM-generated answers and documents can themselves carry formal copies of key claims alongside prose. Data media already carry informal instructions of use (schemas, READMEs); EG-VAR just requires them formal. When upstream providers themselves run EG-VAR-like pipelines, formalization becomes mechanized rather than handcrafted, and data arrives already bearing kernel-checked VERIFIED instructions. The formalization burden moves out of the per-query inference loop into reusable source-side infrastructure; the kernel boundary continues to enforce that only attested claims are labeled VERIFIED.

Governance translation. The two safety theorems (Section 3.6), combined with the no-upcast grade discipline, yield three structural governance properties: (i) tool-attestation-as-axiom enables independent audit of every verified output; (ii) grade discipline makes uncertainty typed and queryable rather than a hidden confidence number; (iii) honest abstention (a kernel rejection becomes the user-visible outcome) is a measurable property, not a learned behaviour. We unpack each in Section 5.2.

2. Related Work

We compare EG-VAR against four buckets of prior work (Table 1): empirical studies of how LLMs trade off parametric priors against external context; symbolic factual verifiers; LLM-driven theorem provers; and execution-attestation systems. Each contributes part of the trust pipeline, but none combines a proof-assistant kernel, tool-attestation as a typed axiom, and honest abstention as a structural property for empirical claims.

The closest empirical prior is ClashEval (Wu et al., 2024) (prior-vs-context conflict on frontier models across six domains; our counterfactual stress tests in Section 4.2 reproduce its prior-retention ranking). Adjacent substrates: symbolic factual verification (Aly & Vlachos, 2024; Krishna et al., 2022; Pei et al., 2025) with learned/shallow checkers and silent-wrong failure modes; post-hoc fact-checking (Chern et al., 2023; Gao et al., 2023; Wei et al., 2024; Dhuliawala et al., 2023); commercial policy guardrails (Amazon Web Services, 2024); LLM-for-math theorem proving (Google DeepMind, 2024; Xin et al., 2025; Achim et al., 2025; Varambally et al., 2025) (same trust model, self-contained claims, no external grounding); autoformalization (Zhou et al., 2024); long-lineage CS verification (Raggi et al., 2016; Bundy, 1988; Clarke et al., 2003; Lahiri et al., 2013); and hardware-attested AI verification (Schnabl et al., 2025; Grigor et al., 2025b;a) (attests execution provenance, not claim truth; complementary to EG-VAR). Detailed treatment of each thread, including the axes EG-VAR extends over ClashEval and why self-contained-math provers do not cover empirical claims, is in Section A.

3. EG-VAR

3.1. Four-layer stack and trust ledger

EG-VAR partitions the agentic-AI loop into four layers with a strict trust ledger (Figure 1; Threat-model summary box below).

L1, Tool layer (trusted, deterministic). The tool layer exposes a fixed set of queries against a source: cell lookups, filtered aggregates, and selection operators on tables, but the pattern is source-agnostic. Each query produces an attested payload: the deterministic answer the tool computed, packaged with a runtime `Attested_T` witness that records query, source identity, and result. The runtime is the only entity that can produce `Attested_T` witnesses.

L2, Per-source formalization (audited offline). Tools attest *storage facts* (e.g., "cell at row r, column c, has value v"); claims live at the world-ontology level (`HasProperty`, `ArgmaxWhere`, `SumWhere`, ...). The mapping from L1 storage facts to L2 world facts is a curator's interpretive commitment (in this paper, an audited proposer–critic–kernel pipeline; Section G.2), encoded as a per-source list of typed *lifts* that the kernel imports as axioms (Section 3.5). The L1/L2 split keeps the formal language describing the world rather than the storage vehicle, the same separation Codd (1970) drew between a relational schema and the predicates it serves.

Threat model (summary). Safety theorems (Theorems 3.1 and 3.2) assume: **Trusted:** Lean 4 kernel, EG-VAR prelude, audited per-source lifts $\Lambda(s)$, deterministic Python runtime. **Untrusted:** LLM analyst/formalizer/solver; they may hallucinate but cannot introduce axioms. **Out of scope:** (i) semantic faithfulness of τ to the NL claim; (ii) a semantically wrong audited lift; (iii) source-data corruption that passes the runtime identity-cell validator. Items (i)–(ii) detailed in Section 5.1.

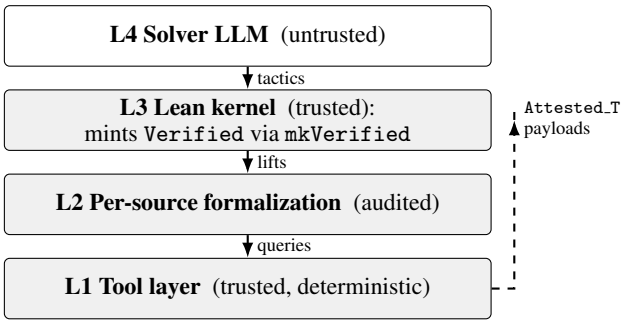

Figure 1. EG-VAR four-layer architecture. Solid arrows: control flow. Dashed arrow: `Attested_T` payloads, the sole load-bearing hypothesis of `mkVerified` (Section 3.4).

L3, Lean 4 kernel (trusted, formal). Lean type-checks every proof step. The kernel is the only entity authorized to mint `Evidence Verified` w for any world-claim w. There is exactly one mint rule, `mkVerified`, which requires an `Attested_T` hypothesis (Section 3.4).

L4, Solver LLM (untrusted, stochastic). The LLM receives a typed goal and proposes proof tactics. The kernel rejects invalid tactics and the rejection messages drive revision. The solver cannot synthesize `Attested_T` terms; it can only consume those introduced by tool calls during the loop.

Trust ledger. The kernel, the L1 vocabulary, the per-source lifts, and the tool adapter are auditable artifacts. The solver is type-checked, not trusted: an arbitrarily wrong solver can never produce a verified-but-unsupported output. This is a *structural* guarantee, not a statistical one. Each verified output is accompanied by the generated Lean proof file, a self-contained replay artifact that any third party with the kernel and the per-source lifts can re-typecheck.

System	Substrate	Evidence input	Checker	Residual risk
TabVer (Aly & Vlachos, 2024)	Nat. logic + arith.	Table cells	Learned NatOp	Silent wrong
ProoFVer (Krishna et al., 2022)	Natural logic	Premises	Shallow symbolic	Silent wrong
FoVer (Pei et al., 2025)	FOL + Z3	Indirect (CoT)	SMT	Consistency $\not\Rightarrow$ grounded
AWS Bedrock (Amazon Web Services, 2024)	Undisclosed	Policy text	Proprietary	Undisclosed
AlphaProof (Google DeepMind, 2024)	Lean 4 (math)	None	Lean kernel	—
DeepSeek-Prover (Xin et al., 2025)	Lean 4 (math)	None	Lean kernel	—
Attestable Audits (Schnabl et al., 2025)	TEE crypto	Exec. bytes	TEE attestation	Wrong-but-attested
Proofs of Autonomy (Grigor et al., 2025b)	MPC-TLS / SNARK	Run provenance	Crypto proof	Wrong-but-attested
EG-VAR (ours)	**Lean 4 + tool attest.**	**Attested L1→L2 lifts**	**Lean kernel**	**Honest abstention**

Table 1. Prior work by substrate, evidence input (what the checker consumes to verify empirical claims), checker, and residual risk under EG-VAR's threat model. EG-VAR is the only row combining a proof-assistant kernel with tool-attested grounding and honest abstention on rejection.

3.2. Tables as formal objects

The deeper move in EG-VAR is on the source side. Empirical verification fails not because claims are unformalized, but because the *source's semantics* are implicit in the tool adapter, hidden in fuzzy retrieval, or assumed in prompt scaffolding. The source is a co-equal trust artifact: from the kernel's perspective a table is an object that needs formalization on the same footing as a claim.

Concretely, three things are formalized per source: (i) an **L1 storage vocabulary** (`CellAt`, `ColumnContains`, `RowMatches`, ...) over which tools emit attested payloads; (ii) an **L2 world ontology** (`HasProperty`, `ArgmaxWhere`, `SumWhere`, ...) over which claims are stated; (iii) a **per-source lift catalog** (Section 3.5) encoding the curator's interpretive commitment about which L1 facts attest which L2 claims under this table's chosen semantics. The trust artifact a third party must inspect is the lift catalog; tools carry L1 facts deterministically, the kernel reasons only over L2 evidence. This separates source-side curation from claim-side reasoning, making both auditable independently.

A dataset admits multiple consistent formalizations (e.g., a matchup table as rank-2 curried vs rank-1 with composite-domain); these are type-isomorphic in any Cartesian closed setting and the kernel admits any consistent one. The curator picks an interpretation and audits the lift menu against the table; pluralism is architectural. Full gauge-freedom argument in Section K.2.

We trade internal completeness for soundness, decidability, and proof economy: L2 predicate equivalences (e.g. MeanWhere $\Leftrightarrow$ SumWhere/CountWhere) are localized in per-source lifts rather than added as kernel lemmas, keeping proof search decidable and traces inspectable. This mirrors SMT theory combination, description logics, and decidable FOL fragments; detailed rationale in Section N.

3.3. Evidence grades and downcast-only

Evidence carries an explicit grade $g \in \{\text{VERIFIED}, \text{SUPPORTED}, \text{PLAUSIBLE}, \text{SPECULATIVE}\}$ ordered by reliability. The kernel admits *downcasting* only ($g_1 \succeq g_2$ and e : `Evidence` g_1 $w \Rightarrow$ `downcast` e : `Evidence` g_2 w). No upcast rule exists, so conclusions cannot outrank their weakest premise. The only constructor for `Evidence Verified _` is `mkVerified`, described next; lower grades arise from explicit downgrades (e.g., staleness, source-conflict) or LLM-proposed heuristic claims that the calculus admits at PLAUSIBLE or SPECULATIVE only.

3.4. The `mkVerified` rule (sole minter of VERIFIED)

Informally: verified evidence exists only if the runtime already supplied an attested payload. The kernel's mint rule is:

```
mkVerified :
  forall (T : ToolId) (q : Query_T)
         (p : Payload_T q)
         (att : Attested_T q p) (w : WProp),
  w in claims (interp_T q p) ->
  Evidence Verified w
```

The hypothesis `att : Attested_T q p` is the load-bearing element. `Attested_T` witnesses are introduced into the proof context only by the runtime in response to a tool call; the solver cannot synthesize them. `interp_T q p` is the per-tool interpretation function (a small total Lean function) that maps the attested payload to a finite set of L2 world claims; the kernel mints `Verified` for any w in that set. There is no other path to `Evidence Verified _`.

3.5. Per-source formalization as the curator's interpretive commitment

The interesting design problem is the L1→L2 bridge. A tool attests "cell at row r, column "state", has value "alabama" " (an L1 storage fact); the claim under verification is "Alabama has the highest HIV incidence rate" (an L2 world fact). The bridge is a per-source list of typed lifts, audited offline by the source's curator, that the kernel imports as

axioms.

Per-source lifts operationalize the L1→L2 bridge: typed composition of an attested L1 fact with a audited lift axiom mints an `Evidence Verified` witness for the matching L2 world-property. Absence of a matching lift is a feature: the solver has no L2 proof path, the kernel rejects, and the system abstains rather than silently guess. The L1/L2 separation is source-agnostic (SQL views, knowledge graphs, and sensor streams admit the same pattern; Section 5.2); a complete worked example (HIV argmax via state projection, showing the lift, attested payload, and Sigma-witness extraction) is in Section H.1.

3.6. Operational proof object and safety theorems

The safety guarantees are stated about the runtime's operational *proof object*, not about Lean's type theory alone. A proof object is the tuple of artifacts that the Python implementation at `src/python/runtime/pipeline.py` actually returns from `run_pipeline` as the fields of `PipelineResult`:

$$\mathrm{ProofObject}(s, c) \;=\; \langle\, \mathit{leanFile},\ \mathit{evidence},\ \mathit{steps} \,\rangle$$

where *leanFile* is the generated Lean module text, *evidence* is the time-ordered list of tool-call payloads lifted to observation-leaf axioms, and *steps* is the history of solver actions and kernel-verbatim feedback. Figure 2 gives a concrete end-to-end instance for one claim. A standalone replay-audit script (`src/python/runtime/verify_axioms.py`, Section G) takes *leanFile* as input and recomputes the whitelist check post-hoc; the script is not invoked in the `run_pipeline` loop. The primary safety gate at generation time is the Lean kernel invocation `lake env lean` inside `pipeline.py`.

Trust zones. Four roles participate in producing the proof object. **Untrusted proposers**: the LLM analyst, formalizer, and solver; none can introduce new top-level axioms because their output is constrained to dictionary entries, a proposed goal type, and a tactic body inserted inside a pre-written Sigma type declaration. **Audited runtime-orchestrator**: the deterministic Python layer that executes tools, emits observation-leaf axioms, assembles the Lean module, invokes the kernel, and records *evidence* and *steps*. **Trusted checker**: the Lean 4 kernel. **Trusted domain assumptions**: the fixed EG-VAR prelude (`EGVar/Basic.lean`, `EGVar/TableL1.lean`, `EGVar/WorldOntology.lean`, and `EGVar/Tactics.lean`) and the audited per-source lifts $\Lambda(s)$ from `data/tier1_table_formalizations.json`, committed at curation time per Section 3.6.

Whitelist-compliant module (summary). We call *leanFile whitelist-compliant for source s* iff every axiom in the dependency closure of the claim proof belongs to one of four buckets: the fixed EG-VAR prelude (**B1**), standard Lean 4 core axioms (**B2**), runtime-emitted observation leaves (**B3**), or per-source lifts matching the regex `lift_[0-9a-f]+_.+` drawn from $\Lambda(s)$ (**B4**); the closure contains no `sorry` variant and at least one trust-artifact witness. Full bucket schemas, the replay-audit mechanics, and guarantee-boundary caveats are in Section P.1.

Algorithm 1 RUNTIMEVERIFY: operational definition of the runtime loop. Trust-zone labels annotate each step. The Python implementation at `src/python/runtime/pipeline.py` realizes this algorithm.

Input: claim c, source s
Output: proof object and verdict
guide ← ANALYST(s, c) *// untrusted LLM*
goalType ← FORMALIZER$(c, \mathit{guide}, \mathit{dict}, \Lambda(s))$
evidence ← $[\,]$; *steps* ← $[\,]$
for $i = 1$ **to** *maxSteps* **do**
  action ← SOLVER$(c, \mathit{goalType}, \mathit{evidence})$
  if *action* is a tool call (t, q) **then**
    payload ← EXECUTE(t, q) *// audited tool*
    if *payload* succeeded **then**
      append $(t, q, \mathit{payload}, \Lambda(s).\mathit{lift})$ to *evidence*
    end if
  else if *action* is a proof attempt *tac* **then**
    leanFile ← BUILD$(\mathit{goalType}, \Lambda(s), \mathit{evidence}, \mathit{tac})$
    $(\mathit{ok}, \mathit{out})$ ← KERNELCHECK$(\mathit{leanFile})$ *// trusted Lean kernel*
    if *ok* **then**
      return $(\mathit{leanFile}, \mathit{evidence}, \mathit{steps})$, VERIFIED
    end if
  end if
  append $(\mathit{action}, \mathit{feedback})$ to *steps*
end for
return $\perp$, ABSTAIN

Theorem 3.1 (No unsupported VERIFIED outputs)**.** *If* RUNTIMEVERIFY*(s, c) returns $(\mathit{leanFile}, \mathit{evidence}, \mathit{steps})$ with verdict* VERIFIED*, and leanFile is whitelist-compliant for s, then every runtime-emitted observation-leaf axiom in the dependency closure of the claim proof is backed by a prior tool execution recorded in evidence, and every remaining axiom dependency lies in the fixed EG-VAR prelude, in the standard Lean 4 core axioms, or in $\Lambda(s)$.*

Theorem 3.2 (No deductive errors)**.** *Every inference step that the system accepts as part of a* VERIFIED *output type-checks in Lean 4's kernel under the declared axiom set.*

Full proofs (checker-fidelity, emission-discipline, and reserved-name lemmas; complete proofs of Theorems 3.1 and 3.2) plus the emission-site table are in Sections P and P.1. The kernel call inside `run_pipeline` (`lake env lean`) is

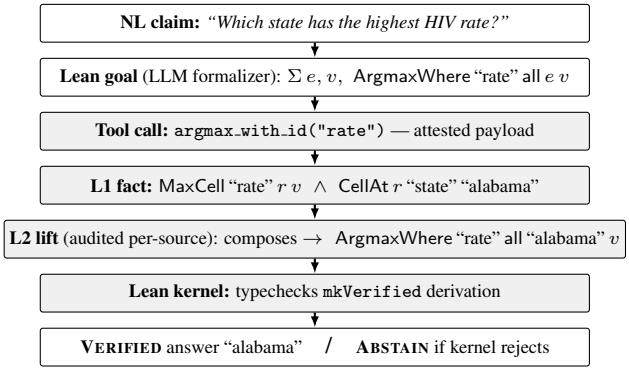

Figure 2. Worked trace for one claim. Untrusted LLM output (white) commits to a goal type; trusted runtime (gray) produces the attested payload, composes it through the audited per-source lift, and the Lean kernel mints VERIFIED iff the derivation type-checks. The Sigma-witness of the accepted proof is the published answer.

the primary safety gate at generation time; the standalone `verify_axioms.py` is a post-hoc replay audit a third-party can re-run offline (Sections G and P.1). Worked examples of attested proofs and per-source lifts are in Sections H and I.

Formalization provenance. Tier 1 and Tier 1.5 load fixture-committed gold types τ^{gold} and audited per-source lifts $\Lambda(s)$ from a frozen fixture (proposer–critic–kernel pipeline, no formalizer invoked at evaluation); Tier 2 invokes an LLM formalizer with access to $\Lambda(s)$ but no gold type. Full provenance (subagent pipeline, human meta-supervision scope, curator yield, collusion firewall) is in Section G.2.

4. Empirical Evaluation

We report three evaluations. The first two load fixture-committed gold goal types, so they isolate the kernelized trust architecture under ideal formalization: Section 4.1 (Tier 1) establishes competitiveness with same-tool baselines on unambiguous claims; Section 4.2 (Tier 1.5) establishes source-faithfulness under counterfactual prior conflict, where the same-tool baseline on Sonnet drops to 80–90% (no-tool 50–80%) while EG-VAR remains 100% across both flip-magnitude regimes and both model strengths. The third evaluation (Section 4.3, Tier 2) removes the gold-fixture bypass and tests the deployment path where the LLM itself proposes the formal goal.

Two axes. We separate *formal safety* (the VERIFIED label is kernel-accepted against the committed trust artifacts; Theorem 3.1) from *semantic faithfulness* (the formalized goal τ matches the natural-language claim; measured by `answer_match` against frozen gold answers). Tiers 1 and 1.5 load $\tau = \tau^{\text{gold}}$ from the frozen fixture, so semantic faithfulness holds by construction; Tier 2 invokes the LLM formalizer, and semantic faithfulness becomes the primary empirical question.

Rung	Tools	Per-source artifact
Table-only	none	— (table only)
Tools-open	full surface	none enforced
Tools-curated	menu-enforced	per-source menu
EG-VAR	menu-enforced	menu + Lean goal + lifts

Table 2. Four-rung ladder isolating tool contribution (Table-only → Tools-open), per-source menu (Tools-open → Tools-curated), and kernelized fixed-goal discipline (Tools-curated → EG-VAR).

Setup. Tier 1 and Tier 1.5 share four *rungs* (Table 2): Table-only (single-shot, no tools); Tools-open (LLM with full runtime tool surface, no per-source menu enforcement); Tools-curated (per-source menu enforced; same tool surface as EG-VAR); EG-VAR (kernel-checked, fixed-goal mode with the curator's typed goal). All rungs use the same model, prompt structure, and answer judge (`answers_match`); EG-VAR additionally requires that the kernel mints a `Verified` witness whose Sigma extract agrees with the gold answer. We evaluate two models: `claude-sonnet-4-6` and `claude-haiku-4-5` (a stronger and a weaker model from the same family). All runs use temperature 0; EG-VAR is deterministic at temperature 0 and we report 1-rep smoke evaluations as canonical for EG-VAR with corroborating multi-rep partial runs (provenance manifest in Section G).

Rung	Match
Table-only	107/120 (89.2%)
Tools-open	113/120 (94.2%)
Tools-curated	114/120 (95.0%)
EG-VAR	**120/120 (100.0%)**

Table 3. Tier 1 ladder, n=120 unambiguous TableBench claims, `claude-sonnet-4-6`, temperature 0.

4.1. Tier 1 gold-goal benchmark: baseline ladder

Our LLM-based curator pipeline selects n=120 unambiguous TableBench claims spanning argmax, sum, count, top-k, and aggregation shapes; for each claim it produces the typed Lean goal and the per-source lifts (the trust artifact EG-VAR consumes; Section G.2). All four rungs are evaluated on Sonnet at temperature 0. Table 3 reports the headline ladder. EG-VAR attains 120/120 (100%) joint formal safety and semantic faithfulness on the gold-formalization path: every claim's kernel proof passes the whitelist replay (Section G, formal safety axis) *and* the extracted Sigma witness matches the curator gold (semantic-faithfulness axis). The same-tool same-model baseline (Tools-curated) plateaus at 114/120 (95%) on the latter alone (no formal-safety surface: baselines cannot abstain); removing the per-source menu

Panel	Rung	Sonnet	Haiku
extreme-flip	Table-only	16/20	15/20
	Tools-open	16/20	20/20
	Tools-curated	16/20	20/20
	EG-VAR	**20/20**	**20/20**
subtle-flip	Table-only	10/20	8/20
	Tools-open	18/20	20/20
	Tools-curated	18/20	20/20
	EG-VAR	**20/20**	**20/20**

Table 4. Tier 1.5 counterfactual ladder: source-faithful count on binary-compare and direction claims, n=20 per panel (temp 0, 1 rep). Residual = prior-override (no wrong-other cases).

costs one further claim (Tools-open 113/120); single-shot table-only floor is 107/120 (89.2%).

4.2. Tier 1.5 counterfactual benchmark: source-faithfulness under prior conflict

Tier 1.5 isolates a qualitatively different failure mode the Tier 1 ladder under-counts. We construct synthetic counterfactual table variants: real TableBench tables with one or two cell values overwritten to contradict the model's world-knowledge prior. For each variant we ask a binary-comparison or aggregation claim and classify the rung's output as *source-faithful* (matches the table), *prior-override* (matches the un-injected real-world value), or *wrong-other*. Tier 1.5's *source-faithfulness* column is the semantic-faithfulness axis under counterfactual-injection pressure; EG-VAR's 100% result combines this with formal-safety (kernel replay) that the baselines lack.

Two flip-magnitude regimes. The *extreme-flip panel* uses magnitude-10-plus reversals (e.g., Italy GDP → \$50, Patriots → 0–16, Sinopec → \$50) across 5 domains; the *subtle-flip panel* uses 10–25% reversals (Mont Blanc → 4500 m vs Monte Rosa 4634 m, Madrid GDP 90 vs Berlin 95) on a panel shifted to stable-scalar priors (country populations, EU country area, European city GDP, Forbes top companies, Alpine peak elevations). Each panel is 5 domains × 2 entity-pair injections × 3 claim shapes = 30 claims per panel (20 binary-compare + direction, 10 cell-lookup; the cell-lookup set doesn't trigger prior-override and is excluded here). Table 4 reports the n=20 binary-compare + direction cells; both models, 1 rep (temp 0, deterministic).

The extreme-flip panel narrows the residual same-tool gap to the single most entrenched prior family (country populations). The subtle-flip panel generalizes prior-override across stable-scalar domains; tools rescue most of it, and the kernel closes the rest.

Per-claim robustness axes (authority-cue, model strength, domain coverage) and the same-tool failure-mode taxonomy are in Section E.1.

Category	Sonnet	Opus
Correct	84.2%	87.5%
Ambiguous claim (logged)	4.2%	4.2%
Benchmark gold error (surfaced)	1.7%	1.7%
Semantic formalizer error	**3.3%**	**1.7%**
Honest abstain	2.5%	5.0%
Solver gave up	4.2%	0.0%

Table 5. Tier 2 decomposition, n=120. Category definitions in Section L.

4.3. Tier 2 end-to-end evaluation: LLM formalizer

Tier 1 and Tier 1.5 bypass the NL-to-Lean formalizer step by loading the fixture-committed goal type from a frozen fixture (Section 3.6). Tier 2 removes that bypass: at deployment time, the runtime invokes the LLM formalizer (Section 3.6) to propose `lean_goal_type` per claim, then proceeds with the same analyst-formalizer-solver-kernel loop. This is the fully automated regime; no curator interaction at runtime.

Setup. n=120 claims, same TableBench curated fixture. `claude-sonnet-4-6` at temperature 0. Pipeline: analyst → formalizer → solver. The audited per-source formalization $\Lambda(s)$ is still loaded; only the claim-level goal type is LLM-proposed. We classify every output into one of six mutually-exclusive reporting categories, based on three observables: *type-equality* with the curator's gold goal type (kernel-`rfl` on string-normalized forms), *kernel acceptance*, and *answer-match* against the gold answer (Table 5).

Formal safety versus semantic faithfulness. Tier 2 measures two distinct properties. *Formal safety* is structural and model-independent: every VERIFIED proof still depends on an attested tool leaf, regardless of which LLM proposed the goal type (Theorem 3.1). *Semantic faithfulness* is operational and model-dependent: it measures whether the LLM-proposed goal matches the frozen reading of the claim. Table 5 reports the semantic-faithfulness axis per model; the *Semantic formalizer error* row is the pure formalizer-error rate.

Tier 2 under counterfactual pressure. To check that the LLM-as-formalizer pipeline is not just a curator-fixture artifact, we re-run Tier 2 Sonnet on the Tier 1.5 extreme-flip panel, restricted to binary-compare and direction claims (n=20, prior-override-injected tables). Every claim produces a source-faithful VERIFIED output (**20/20 SF**), with the extracted Sigma witness matching the table-injected answer rather than the model's world-knowledge prior. The Tier 1.5 same-tool baseline drops to 16/20 on the same extreme-flip tables (Section 4.2); Tier 2 with end-to-end LLM formalization matches EG-VAR's gold-formalization rung on this subset.

5. Discussion

5.1. Limitations

Three scope limitations bound the empirical claims. **(i)** Tier 1/1.5 bypass the formalizer by loading fixture-committed gold types from a frozen fixture (Section 3.6); Tier 2 removes that bypass. **(ii)** Tools, per-source lifts, and the L1 adapter are trusted; a semantically wrong audited lift can certify a wrong formalized claim (Section 3.5). **(iii)** All evaluations are on single TableBench tables; multi-source, KG, and streaming sources are out of scope. Detailed scope limits are in Section M.

5.2. Governance translation

Theorems 3.1 and 3.2 and the no-upcast grade discipline yield three governance-relevant structural properties.

(G1) Auditable evidence trails. Every `Verified` output carries the generated Lean proof file as its replay artifact: a third-party auditor with the kernel, per-source lifts, and tool adapter can re-typecheck the claim from the attested payload without re-running the LLM. The proof term is a truthmaker in Fine (2017)'s sense: the specific attested state of affairs that grounds the claim. Composition of evidence across multiple lifts carries the algebraic structure of a provenance semiring (Green et al., 2007). The trust artifact is enumerable (L1 vocabulary, per-source lifts, tool adapter, kernel); audit cost is *amortized*, each per-source lift audited once at curation time and reused across all subsequent claims.

(G2) Typed uncertainty. Grade discipline (no-upcast, minimum-of-premises) makes uncertainty inspectable rather than a hidden confidence number. The graded `Evidence` g w object is operationally close to a justification term (Artemov, 2008), where evidence is explicit rather than folded into a model's posterior. A PLAUSIBLE output cannot be silently re-introduced downstream as VERIFIED; an auditor can query which claims arrived at which grade and why.

(G3) Honest abstention. A kernel rejection surfaces as a user-visible ABSTAIN, not a silent fallback guess. Refusal thresholds are externally specifiable (a downstream policy can require VERIFIED and reject ABSTAIN, or admit ABSTAIN as a flagged state, without re-implementing the trust boundary); the abstention rate is measurable per-source; the failure mode is auditable rather than learned.

Portability beyond tables. The L1/L2/lift pattern is source-agnostic: SQL views, knowledge graphs, and sensor streams each admit the same per-source-formalization treatment under a matching attestation discipline. The empirical cross-substrate panel is itemized in Section 5.3; the broader substrate- adoption argument (formalization gap shrinks as sources become formal; Section K.3).

5.3. Future work scoped to governance reach

Three extensions: (1) fine-tuning the NL formalizer on the curator-provided (claim, gold type) corpus with kernel acceptance as the reward signal, an accuracy intervention that tightens the Tier 2 residual rate without weakening Theorem 3.1; (2) multi-source attestation with conflict-downgrade rules (VERIFIED $\rightarrow$ SUPPORTED on detected disagreement); (3) cross-substrate panel (KG-grounded, SQL-grounded) to test the L1/L2 portability claim (Section 5.2). Detail in Section M.

6. Conclusion

EG-VAR is an architecture for verified empirical claims grounded in attested external data, instantiated on tables as the first end-to-end substrate. Tool access alone does not suffice: frontier LLMs prior-override counterfactual tables on a measurable fraction of claims. EG-VAR addresses this architecturally: the Lean 4 kernel mints VERIFIED claims only under tool attestation and type-checks every accepted proof step. TableBench Tier 1 (n=120) closes the same-tool gap to zero; Tier 1.5 counterfactuals (n=20 per panel, two flip regimes, two models) stay 100% source-faithful while same-tool drops to 80–90% (no-tool 50–80%); end-to-end Tier 2 with LLM formalization bounds formalizer error at 3.3% (Sonnet), 1.7% (Opus).

The core contribution is structural. Theorem 3.1 rules out unsupported VERIFIED outputs model-independently; Theorem 3.2 guarantees that every accepted deduction type-checks under the declared axiom set. Remaining failures are not silent hallucinations but measurable, auditable formalization failures, logged ambiguities, or honest abstentions. This is a *constructive path toward* eliminating LLM hallucination in empirical inference: the safety floor is architectural today; the semantic-formalization residual is an explicit accuracy target addressable by formalizer fine-tuning on the curator corpus, consensus formalization, and longer-term typed sidecars on sources and LLM-generated documents.

For AI governance, three properties become enforceable at the system boundary rather than learned behaviours: auditable evidence trails via replayable per-claim Lean proofs, typed uncertainty under no-upcast grade discipline, and externally specifiable refusal thresholds via honest abstention. Verifiable grounding generalises as an architectural protocol rather than a table-specific trick: the same L1/L2/lift discipline can in principle apply to SQL views, knowledge graphs, typed registries, and other attested sources. Validating beyond tables is the immediate empirical next step; the longer-run bet is that as sources and documents acquire typed sidecars, more of the formalization burden moves from inference loops into reusable infrastructure.

Acknowledgements

I thank the TAIGR reviewers for feedback that sharpened the governance framing and the distinction between unsupported VERIFIED outputs and semantic-formalization failures. JR is partially supported by a Vannevar Bush Faculty Fellowship ONR N000142312863.

References

Abiteboul, S., Buneman, P., and Suciu, D. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

Achim, T. et al. Aristotle: IMO-level automated theorem proving, 2025. https://arxiv.org/abs/2510.01346.

Alchourrón, C. E., Gärdenfors, P., and Makinson, D. On the logic of theory change: partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(2): 510–530, 1985.

Aly, R. and Vlachos, A. TabVer: Tabular fact verification with natural logic. *Transactions of the Association for Computational Linguistics (TACL)*, 2024.

Amazon Web Services. What are automated reasoning checks in amazon bedrock guardrails?, 2024. Documentation, accessed 2026-04-23. https://docs.aws.amazon.com/bedrock/latest/userguide/guardrails-automated-reasoning-checks.html.

Anderson, A. R. and Belnap, N. D. *Entailment: The Logic of Relevance and Necessity*, volume 1. Princeton University Press, 1975.

Artemov, S. The logic of justification. *The Review of Symbolic Logic*, 1(4):477–513, 2008.

Barwise, J. and Perry, J. *Situations and Attitudes*. MIT Press, 1983.

Belnap, N. D. A useful four-valued logic. *Modern Uses of Multiple-Valued Logic*, pp. 5–37, 1977.

Bundy, A. The use of explicit plans to guide inductive proofs. In *Proceedings of the 9th International Conference on Automated Deduction (CADE-9)*, volume 310 of *Lecture Notes in Computer Science*, pp. 111–120. Springer, 1988.

Burgess, J. P. and Rosen, G. *A Subject With No Object: Strategies for Nominalistic Interpretation of Mathematics*. Oxford University Press, 1997.

Caminada, M. and Amgoud, L. On the evaluation of argumentation formalisms. *Artificial Intelligence*, 171(5-6): 286–310, 2007.

Chern, I.-C. et al. FacTool: Factuality detection in generative AI — a tool augmented framework for multi-task and multi-domain scenarios, 2023. https://arxiv.org/abs/2307.13528.

Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

Cooper, R. Type theory and semantics in flux. In *Handbook of the Philosophy of Science*, 2012.

de Moura, L. and Ullrich, S. The Lean 4 theorem prover and programming language. In *Proceedings of the 28th International Conference on Automated Deduction (CADE-28)*. Springer, 2021. https://doi.org/10.1007/978-3-030-79876-5_37.

Dhuliawala, S. et al. Chain-of-verification reduces hallucination in large language models. 2023. https://arxiv.org/abs/2309.11495.

Donnellan, K. S. Reference and definite descriptions. *The Philosophical Review*, 75(3):281–304, 1966.

Doyle, J. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, 1979.

Fine, K. Truthmaker semantics. In Hale, B., Wright, C., and Miller, A. (eds.), *A Companion to the Philosophy of Language*, pp. 556–577. Wiley-Blackwell, 2017.

Frege, G. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, 100:25–50, 1892.

Gao, L. et al. RARR: Researching and revising what language models say, using language models. In *Proceedings of ACL*, 2023.

Goldman, A. What is justified belief? *Justification and Knowledge*, pp. 1–23, 1979.

Google DeepMind. AI achieves silver-medal standard solving International Mathematical Olympiad problems, 2024. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/.

Green, T. J., Karvounarakis, G., and Tannen, V. Provenance semirings. In *PODS*, pp. 31–40, 2007.

Grigor, A., Schroeder de Witt, C., Birnbach, S., and Martinovic, I. VET your agent: Towards host-independent autonomy via verifiable execution traces, 2025a. https://arxiv.org/abs/2512.15892.

Grigor, A., Schroeder de Witt, C., and Martinovic, I. Proofs of autonomy: Scalable and practical verification of AI autonomy. In *Workshop on Technical AI Governance (TAIG@ICML)*, 2025b. OpenReview qxFgQHN69d.

Howard, W. A. The formulae-as-types notion of construction. In Seldin, J. P. and Hindley, J. R. (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pp. 479–490. Academic Press, 1980.

Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., and Scarfone, K. Guide to attribute based access control (ABAC) definition and considerations. Technical report, NIST SP 800-162, 2014.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., and Liu, T. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 2025. https://arxiv.org/abs/2311.05232.

Jøsang, A. *Subjective Logic: A Formalism for Reasoning Under Uncertainty*. Springer, 2016.

Kripke, S. A. *Naming and Necessity*. Harvard University Press, 1980.

Krishna, A., Riedel, S., and Vlachos, A. ProofVer: Natural logic theorem proving for fact verification. In *Transactions of the Association for Computational Linguistics (TACL)*, 2022. https://aclanthology.org/2022.tacl-1.59/.

Lahiri, S. K., McMillan, K. L., Sharma, R., and Hawblitzel, C. Differential assertion checking. In *ESEC/FSE*, 2013.

Li, J., Yu, L., and Ettinger, A. Counterfactual reasoning: Testing language models' understanding of hypothetical scenarios. In *Proceedings of ACL (short)*, 2023. https://aclanthology.org/2023.acl-short.70/.

Mac Lane, S. *Categories for the Working Mathematician*. Springer, 1971.

Mallen, A., Asai, A., Zhong, V., Das, R., Khashabi, D., and Hajishirzi, H. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of ACL*, 2023. https://arxiv.org/abs/2212.10511.

Marjanovic, S. V. et al. DynamicQA: Tracing internal knowledge conflicts in language models. In *Findings of EMNLP*, 2024. https://arxiv.org/abs/2407.17023.

Martin-Löf, P. *Intuitionistic type theory*. Bibliopolis, Naples, 1984.

Ming, Y. et al. FaithEval: Can your language model stay faithful to context, even if "The Moon is Made of Marshmallows"? In *Proceedings of ICLR*, 2025. https://arxiv.org/abs/2410.03727.

Montague, R. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language*, pp. 221–242. 1973.

Pei, Y., Du, Y., and Jin, X. FoVer: First-order logic verification for natural language reasoning. *Transactions of the Association for Computational Linguistics*, 13:1340–1359, 2025. https://aclanthology.org/2025.tacl-1.61/.

Quine, W. V. O. *Ontological Relativity and Other Essays*. Columbia University Press, 1969.

Raggi, D., Bundy, A., Grov, G., and Pease, A. Automating change of representation for proofs in discrete mathematics (extended version). *Mathematics in Computer Science*, 10(4):429–457, 2016. https://doi.org/10.1007/s11786-016-0275-z.

Schnabl, C., Hugenroth, D., Marino, B., and Beresford, A. R. Attestable audits: Verifiable AI safety benchmarks using trusted execution environments. In *Workshop on Technical AI Governance (TAIG@ICML)*, 2025. https://arxiv.org/abs/2506.23706.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. https://homotopytypetheory.org/book/.

Varambally, S., Voice, T., Sun, Y., Chen, Z., Yu, R., and Ye, K. Hilbert: Recursively building formal proofs with informal reasoning, 2025. https://arxiv.org/abs/2509.22819.

Wei, J. et al. Long-form factuality in large language models (SAFE). 2024. https://arxiv.org/abs/2403.18802.

Wu, K., Wu, E., and Zou, J. ClashEval: Quantifying the tug-of-war between an LLM's internal prior and external evidence. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024. https://arxiv.org/abs/2404.10198.

Wu, X., Yang, J., Chai, L., Zhang, G., Liu, J., Du, X., et al. TableBench: A comprehensive and complex benchmark for table question answering. In *Proceedings of AAAI*, 2025. https://arxiv.org/abs/2408.09174.

Xie, J., Zhang, K., Chen, J., Lou, R., and Su, Y. Adaptive chameleon or stubborn sloth: Revealing the behavior of large language models in knowledge conflicts. In

Proceedings of ICLR, 2024. https://arxiv.org/abs/2305.13300.

Xin, H. et al. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *Proceedings of ICLR*, 2025. https://arxiv.org/abs/2408.08152.

Zhou, J. P., Staats, C., Li, W., Szegedy, C., Weinberger, K. Q., and Wu, Y. Don't trust: Verify — grounding LLM quantitative reasoning with autoformalization. In *ICLR*, 2024. arXiv:2403.18120.

A. Extended related work

Prior-vs-source conflict in LLMs. The closest empirical prior is ClashEval (Wu et al., 2024), which evaluates frontier models on prior-vs-context conflict across six domains. They report >60% prior-override on incorrect retrieved content and that Claude scores highest on prior retention among the frontier families evaluated (Context-Bias 0.157 for Opus, 0.201 for Sonnet, 0.245 for Gemini 1.5, 0.304 for GPT-4o). Our counterfactual stress tests reproduce this ranking quantitatively for Sonnet (Section 4.2). We extend the analysis along two axes prior work has not separated: (i) *authority-cue robustness* — explicit "according to the table" framing rescues some claim shapes but not the strongest prior-bound ones; and (ii) *flip-magnitude regime* — subtle plausible flips trigger prior-override on 4 of 5 stable-scalar domains where extreme flips trigger only one, suggesting the model treats outrageous tool outputs as "alternate scenarios." This disambiguates *dynamicity* (Marjanovic et al., 2024) from *popularity* (Mallen et al., 2023) in the trigger pattern, and is consistent with Li et al. (2023)'s finding that counterfactual behavior depends on interactions between world knowledge and contextual cues.

Symbolic factual verification. TabVer (Aly & Vlachos, 2024) and ProoFVer (Krishna et al., 2022) each add a verification step over LLM outputs using natural logic; the verifier is itself a learned or shallow-symbolic system that produces silent-wrong outputs on classifier or premise-mapping error. FoVer (Pei et al., 2025) translates chain-of-thought into first-order logic checked by Z3, but verifies *reasoning consistency* rather than empirical grounding — a CoT can be internally consistent yet factually disconnected from the source. Post-hoc fact-checking pipelines such as FacTool (Chern et al., 2023), RARR (Gao et al., 2023), SAFE (Wei et al., 2024), and CoVe (Dhuliawala et al., 2023) attempt detection after generation rather than structural grounding during generation; they catch some hallucinations but cannot guarantee abstention on ungrounded outputs. Commercial guardrails such as AWS Bedrock's automated reasoning checks (Amazon Web Services, 2024) occupy a related space at the policy-rule layer. EG-VAR differs by replacing the learned/shallow checker with a proof-assistant kernel and a typed evidence calculus, structurally translating kernel rejection into user-visible abstention.

Proof-assistant theorem proving (math). A separate large literature applies LLMs to mathematical theorem proving via Lean, Coq, or Isabelle: AlphaProof (Google DeepMind, 2024), Aristotle (Achim et al., 2025), DeepSeek-Prover (Xin et al., 2025), Hilbert (Varambally et al., 2025). These systems share our trust model (LLM untrusted, kernel trusted) but operate on *self-contained* mathematical claims — proof obligations derivable from the calculus alone, with no external grounding. The novel architectural commitment in EG-VAR is the `mkVerified` mint rule (Section 3.4), which requires an `Attested_T` from the tool runtime as a hypothesis — extending the proof-assistant trust model to empirical claims grounded in deterministic external sources. Autoformalization systems such as DTV (Zhou et al., 2024) use a similar LLM-as-formalizer pattern; Tier 1 and Tier 1.5 bypass the deployment-time formalizer via fixture-committed gold formalizations generated by the proposer–critic–kernel pipeline, isolating the kernel-discipline contribution.

Lineage in CS verification. The systems-level concerns that shape EG-VAR — separating semantic specification from proof strategy, refusing to weaken the specification when the proof gets hard, and inheriting kernel-checked discipline from a substrate designed for harder claims — are old questions in the verification literature. Raggi et al. (2016) formalize that semantically equivalent encodings can have very different proof difficulty, motivating our refusal to add cross-predicate equivalence lemmas in the kernel. Bundy (1988) introduced explicit proof planning, anticipating our formalizer/solver information barrier. CEGAR (Clarke et al., 2003) established the discipline of refining the verification plan, not the semantic target, when proofs fail — the same discipline our masked-table architecture enforces on the formalizer. Verification-aware specification languages (Lahiri et al., 2013) address the same informal-intent to formal-specification gap that our per-source formalization protocol addresses for empirical sources. EG-VAR inherits these moves and applies them at a different boundary: not program correctness, but LLM-mediated empirical claim verification. The novelty is the transfer and recombination, not the invention of the underlying discipline.

Hardware-attested AI verification. Attestable Audits (Schnabl et al., 2025) use Trusted Execution Environments to emit remote-attestation signatures binding $Hash(\text{model})$, $Hash(\text{audit code})$, and a benchmark digest, proving that approved bytes executed on approved inputs. Proofs of Autonomy (Grigor et al., 2025b) use MPC-TLS, SNARKs, or TEEs to prove with negligible forgery probability that a given output trace was produced by a configured agent rather than a malicious host (see also Grigor et al., 2025a). Both verify execution provenance — *who* ran the computation, on *what* bytes — not whether the computation's claims are true. EG-VAR is complementary: we typecheck a deductive derivation of each empirical claim from attested tool payloads, ruling out verified hallucinations at the logical level rather than the cryptographic level. An EG-VAR runtime could in principle run inside such an enclave for stack composition.

B. Per-claim diff for the Tier 1 ladder

The supplement includes `scripts/tier1/build_baseline_ladder_csv.py`, which generates `baseline_ladder_per_claim.csv` from the four E1 rung JSONLs: one row per Tier 1 claim with the goal type, gold answer, and per-rung (Table-only / Tools-open / Tools-curated / EG-VAR) status, answer string, and match flag. The 6 claims that distinguish EG-VAR from Tools-curated are flagged in the output.

C. The EG-VAR Lean 4 library

The EG-VAR prelude is shipped as a self-contained Lean 4 library at `src/lean/` with a Lake build (`lakefile.toml`). The library is the kernel-trusted portion of the trust ledger (Section 3.1): every generated claim module imports it verbatim, and Theorem 3.1's whitelist-bucket **B1** refers to its declarations. This appendix documents the module layout, the extension surface, and the third-party replay-audit contract.

Module layout. The library root `EGVar.lean` re-exports eight modules:

- `EGVar.Basic` — sorts and kernel mint rule: the abstract `WProp` world-proposition sort, the `Grade` inductive (`Speculative` $\prec$ `Plausible` $\prec$ `Supported` $\prec$ `Verified`) with meet/join lattice laws, the `Evidence` family, `Attested_T`/`ClaimTag_T`/`claimWProp`, and the sole `Verified`-minter `mkVerified`. Contains `downcast` (the only grade-weakening step) and the common-sense `CSAttestation` / `mkPlausible` path.
- `EGVar.Safety` — the structural lemmas used in Section P (evidence-closure discipline, downcast non-upgrade).
- `EGVar.TableL1` — L1 storage-vocabulary shells: `RowHandle`, `CellAt`, `Cond` constructors (`all`, `propEq`, `propContains`, `propGte`, `and`, `or`), and the L1 predicate shells (`SumCells`, `CountCells`, `CountDistinctCells`, `MeanCells`, `MaxCellValue`/`MinCellValue`, `MaxCell`/`MinCell`, `TopK*Cells`, `BottomK*Cells`).
- `EGVar.WorldOntology` — L2 world-property predicates: `HasProperty`, `Value` (with `asRat`, `ValueGt`), and the L2 predicates that per-source lifts target (`SumWhere`, `CountWhere`, `MeanWhere`, `MaxWhere`/`MinWhere`, `ArgmaxWhere`/`ArgminWhere`, `TopK*Where`, `BottomK*Where`). This module defines the vocabulary in which $\Lambda(s)$ entries are phrased.
- `EGVar.WorldRules` — structural world-proposition combinators (`PLift` propositional lifting, `PProd` conjunction as Sigma-tail, sigma-binding utilities).
- `EGVar.Tactics` — proof-tactic helpers used by the solver (Sigma-witness `refine` patterns, `exact` sugar for L2 discharge).
- `EGVar.Ontology`, `EGVar.Rules` — legacy weather/finance domain axioms kept for backward compatibility with Phase 1 fixtures; not imported by Tier 1/1.5/2 generated modules.

Library build. The Lake manifest declares a single library target `EGVar` and an executable target `egvar` (rooted at `Main.lean`) for interactive smoke-testing. A third-party user runs `lake build` once to compile the library, then `lake env lean <generated_claim>.lean` to elaborate a generated proof against the prelude. The runtime orchestrator invokes exactly this command at `pipeline.py:4428`; the replay-audit script `verify_axioms.py` invokes the same elaboration with additional `#print axioms` processing.

Extension surface. Adding a new tool to EG-VAR reduces to three kernel-level steps, each kernel-checked: (i) declare a new `ToolId` (a pure identifier) and its `Query_T` and `Payload_T` families in a user-supplied module; (ii) extend the L1 predicate menu in the style of `EGVar.TableL1` if the tool exposes a new storage shape; (iii) add the corresponding L2 predicates and per-source lifts targeting them in the style of `EGVar.WorldOntology`. The mint rule `mkVerified` is unchanged: every new tool reuses the same `Evidence` `Verified`-constructor path and cannot weaken Theorem 3.1. Adding a new ontology role-slot (entity or property) is purely data: both are `String` literals under R5/R10 per Section 3.5 and require no kernel changes.

Third-party replay-audit contract. A deployment that publishes EG-VAR-generated answers as trust-bearing artifacts must publish the following bundle for each answer: (a) the generated `.lean` module (the *leanFile* field of the proof object, Section 3.6); (b) the pinned commit hash of the EG-VAR library and of `data/tier1_table_formalizations.json`;

(c) the corresponding entry of the evidence log describing which tool executions backed the observation-leaf axioms. Any third party holding this bundle and a Lean 4 toolchain can re-run the audit by cloning the EG-VAR library at the pinned commit, running `lake build`, then running `python -m runtime.verify_axioms <claim_module>.lean` (which invokes the kernel, reads `#print axioms`, and checks the bucket predicate); no LLM is invoked during replay. We treat this contract as central to the paper's *attestable* claim; the library is designed so that the boundary between trusted and untrusted artifacts is mechanically checkable offline, not contingent on the original generation pipeline being available.

Line counts. The eight modules total ≈1.35 kLOC: `Basic` 192, `Ontology` 210 (legacy), `Rules` 148 (legacy), `Safety` 86, `TableL1` 134, `Tactics` 217, `WorldOntology` 278, `WorldRules` 82. Excluding the two legacy modules the active prelude is ≈1.0 kLOC. This is the full trusted Lean code that a third-party auditor must read to evaluate Theorem 3.1's **B1** bucket; the other three buckets (**B2**–**B4**) are verified by the mechanical `verify_axioms.py` predicate.

C.1. Code listings

We reproduce the key signatures so that the main-text references to `Evidence`, `mkVerified`, `Cond`, `HasProperty`, `ArgmaxWhere`, and `ValueGt` typecheck against the actual kernel definitions imported by every generated proof.

Sorts and the kernel mint rule (`src/lean/EGVar/Basic.lean`)**.**

```
-- World propositions: separate sort from Prop
opaque WProp : Type

-- Evidence grades, ordered Verified > ... > Speculative
inductive Grade where
  | Verified
  | Supported
  | Plausible
  | Speculative

-- Indexed evidence type
opaque Evidence : Grade -> WProp -> Type

-- Sole minter of Evidence Grade.Verified
axiom mkVerified
  {T : ToolId} {q : Query_T T} {p : Payload_T T q}
  (att : Attested_T T q p)
  {w : WProp}
  (mem : ClaimOf w (interp_T T q p)) :
  Evidence Grade.Verified w

-- Downcast: only direction
axiom downcast {g1 g2 : Grade} {w : WProp}
  (h : g1 >= g2) (e : Evidence g1 w) :
  Evidence g2 w
```

World ontology (`src/lean/EGVar/WorldOntology.lean`)**.**

```
-- Value type: numeric via Rat, or string identifier
inductive Value where
  | int  (n : Int)
  | rat  (r : Rat)
  | str  (s : String)

-- Core L2 world facts
axiom HasProperty
  (entity : String) (property : String)
  (value : Value) : WProp

axiom Relation
  (subject : String) (rel : RelName)
  (object : String) : WProp
```

```
-- Aggregate predicates
axiom SumWhere   (col : String) (c : Cond) (v : Value) : WProp
axiom CountWhere (col : String) (c : Cond) (n : Value) : WProp
axiom MeanWhere  (col : String) (c : Cond) (v : Value) : WProp
axiom MaxWhere   (col : String) (c : Cond) (v : Value) : WProp
axiom MinWhere   (col : String) (c : Cond) (v : Value) : WProp

-- Entity-anchored aggregate (level 3 of App H hierarchy)
axiom ArgmaxWhere
  (col : String) (c : Cond)
  (entity : String) (v : Value) : WProp
axiom ArgminWhere
  (col : String) (c : Cond)
  (entity : String) (v : Value) : WProp

-- Decidable Prop-level value comparisons
def ValueGt (a b : Value) : Prop :=
  match a.toRat, b.toRat with
  | some x, some y => x > y
  | _, _ => False
-- (analogous: ValueLt, ValueGte, ValueLte, ValueEq;
-- all carry Decidable instances)

-- Composable filter
inductive Cond where
  | propEq       (col : String) (v : Value) : Cond
  | propGt       (col : String) (v : Value) : Cond
  | propLt       (col : String) (v : Value) : Cond
  | propGte      (col : String) (v : Value) : Cond
  | propLte      (col : String) (v : Value) : Cond
  | propContains (col : String) (v : Value) : Cond
  | and (c1 c2 : Cond) : Cond
  | or  (c1 c2 : Cond) : Cond
  | not (c : Cond)     : Cond
  | all                : Cond
```

L1 storage primitives (`src/lean/EGVar/TableL1.lean`)**.**

```
-- Row-handle (opaque): the per-source lift binds to it
opaque RowHandle : Type

-- L1 facts: tools attest these directly
axiom CellAt
  (r : RowHandle) (col : String) (v : Value) : WProp

axiom MaxCellValue
  (col : String) (c : Cond) (v : Value) : WProp
axiom MinCellValue
  (col : String) (c : Cond) (v : Value) : WProp

-- Row-anchored aggregates: identify which row achieves the max/min
axiom MaxCell
  (col : String) (c : Cond)
  (r : RowHandle) (v : Value) : WProp
axiom MinCell
  (col : String) (c : Cond)
  (r : RowHandle) (v : Value) : WProp
```

Per-source lift example (curated for the lunar-craters table). The lift binds an L1 `MaxCell` witness with an L1 `CellAt` witness on the "name" column to mint an L2 `ArgmaxWhere` claim:

```
axiom craters_lift_argmax_via_name_proj
  (col : String) (c : Cond)
  (r : RowHandle) (name : String) (v : Value) :
```

```
    Evidence Grade.Verified (MaxCell col c r v) ->
    Evidence Grade.Verified
      (CellAt r "name" (Value.str name)) ->
    Evidence Grade.Verified
      (ArgmaxWhere col c name v)
```

This is the curator's interpretive commitment for this table: "name" is the row-identifying column, so an `argmax_with_id` tool result lifts uniquely to an `ArgmaxWhere` claim under any `Cond`. A different table whose row-identifying column is, say, `state` carries its own analogous lift; the two are independent.

D. Per-source lift catalog (countries-population table)

To make "per-source formalization is the trust artifact" concrete, we reproduce the lift catalog the curator pipeline produced for the countries-population table (`04744378...` in TableBench). This is the table that all of no-cue pilot, authority-cued pilot, and subtle-flip panel D1 use, so the catalog underpins the empirical claims in Section 4.2. The full catalog has 63 lifts; we show one representative per L2 predicate cluster.

Curator's interpretation declaration. "Country populations 2013. Identity column = `country (or dependent territory)`. Numeric columns: rank, july 1 2013 projection, % of pop, average relative annual growth (%), average absolute annual growth (all strict numerics)."

Aggregate-only lifts (level 1 in App H hierarchy). One lift per (column, aggregate operation) pair. All have the same shape: L1 column-strict aggregate → L2 `*Where` claim. Examples on the population column:

```
axiom lift_04744378_pop_sum
  (c : Cond) (v : Value) :
    Evidence Grade.Verified
      (SumCells "july 1 , 2013 projection" c v) ->
    Evidence Grade.Verified
      (SumWhere "july 1 , 2013 projection" c v)

axiom lift_04744378_pop_mean
  (c : Cond) (v : Value) :
    Evidence Grade.Verified
      (MeanCells "july 1 , 2013 projection" c v) ->
    Evidence Grade.Verified
      (MeanWhere "july 1 , 2013 projection" c v)

axiom lift_04744378_pop_max
  (c : Cond) (v : Value) :
    Evidence Grade.Verified
      (MaxCellValue "july 1 , 2013 projection" c v) ->
    Evidence Grade.Verified
      (MaxWhere "july 1 , 2013 projection" c v)

axiom lift_04744378_pop_count_where
  (c : Cond) (v : Value) :
    Evidence Grade.Verified
      (CountCells "july 1 , 2013 projection" c v) ->
    Evidence Grade.Verified
      (CountWhere "july 1 , 2013 projection" c v)
```

Analogous lifts exist for `MinWhere`, `TopKSumWhere`, `TopKMeanWhere`, `BottomKSumWhere`, `BottomKMeanWhere`, and `CountDistinctWhere`.

Entity-anchored lifts (level 3 in App H hierarchy). The curator commits that `country (or dependent territory)` is the row-identifying column. Each row-identifying lift takes an L1 row-anchored aggregate (`MaxCell` / `MinCell`) plus an L1 `CellAt` on the identity column to mint an `ArgmaxWhere` or `ArgminWhere` L2 claim.

```
axiom lift_04744378_pop_argmax_via_country
  (col : String) (c : Cond)
```

```
  (r : RowHandle) (name : String) (v : Value) :
    Evidence Grade.Verified (MaxCell col c r v) ->
    Evidence Grade.Verified
      (CellAt r "country (or dependent territory)"
                (Value.str name)) ->
    Evidence Grade.Verified
      (ArgmaxWhere col c name v)

axiom lift_04744378_pop_argmin_via_country
  (col : String) (c : Cond)
  (r : RowHandle) (name : String) (v : Value) :
    Evidence Grade.Verified (MinCell col c r v) ->
    Evidence Grade.Verified
      (CellAt r "country (or dependent territory)"
                (Value.str name)) ->
    Evidence Grade.Verified
      (ArgminWhere col c name v)
```

The argmax/argmin lifts are parametric over the measure column — one lift per identity column suffices to cover argmax claims on any numeric column, which is why the full catalog has 5 entity-anchored entries (one per identity choice), not 5×5.

Cell-lookup lift. A single lift mints `HasProperty` claims by composing two L1 `CellAt` attestations: the entity at row r in the identity column equals e, and the value at row r in the measure column equals v.

```
axiom lift_04744378_pop_cell_lookup
  (e : String) (col : String)
  (r : RowHandle) (v : Value) :
    Evidence Grade.Verified
      (CellAt r "country (or dependent territory)"
                (Value.str e)) ->
    Evidence Grade.Verified (CellAt r col v) ->
    Evidence Grade.Verified
      (HasProperty e col v)
```

What this catalog buys (and what it doesn't). Composition is by lift: any L2 claim using `SumWhere`, `MeanWhere`, `MaxWhere`/`MinWhere`, `ArgmaxWhere`/`ArgminWhere`, `CountWhere`, `TopKSumWhere`, etc. over a numeric column listed above will type-check. A claim using a predicate not in the catalog — say `Relation` (the table has no relational structure) or `ArgmaxWhere` on a non-numeric column — has no L2 path; the solver cannot construct `Evidence Verified` for it, and the system abstains. The catalog is the table's L2 *commitment boundary*: claims inside compose; claims outside abstain. Adding a new claim shape requires adding a new lift (curation surface), not relaxing the kernel discipline.

E. Tier 1.5 per-claim data, ablations, and pre-registration

E.1. Robustness axes and failure taxonomy

Three robustness axes. Three additional ablations strengthen the discriminator (full per-claim breakdowns below): **(i) Authority cue.** On the no-cue pilot countries-population pilot, prefixing every claim with "According to the table below, ..." rescues some claim shapes (yes/no thresholds, 3-way smallest) but the binary compare V3_C1 (Egypt vs. Yemen, table-injected) stays 0/3 source-faithful on both Tools-curated and Tools-open with the cue. **(ii) Model strength.** Haiku 4.5 is fully tool-deferent on Tools rungs (90/90 in every cell); Sonnet 4.6 prior-overrides even with tools (residual concentrated on countries-pop and Madrid-vs-Berlin GDP). The kernel-vs-baseline gap is therefore largest exactly where parametric priors are strongest — consistent with Wu et al. (2024) placing Claude with the highest prior retention among frontier families. **(iii) Domain coverage.** Under extreme flips, Sonnet prior-override concentrates on country populations only (other 4 extreme-flip panel domains 100% SF on Tools-curated). Under subtle flips, the same-tool baseline fails on 4 of 5 subtle-flip panel domains on Table-only — multi-domain replication. Tools rescue most subtle-flip failures on Tools rungs; the kernel closes the residual.

Failure-mode taxonomy. Of the 6/90 Tools-curated failures on subtle-flip panel Sonnet, all are binary-compare prior-overrides; EG-VAR resolves them by minting from the attested cell-lookup pair via a per-source `ArgmaxWhere`-with-

disjunction lift. Cell-lookup and sum-aggregation control claims are 100% source-faithful on every Sonnet rung, including Table-only — consistent with the literature finding that pure information-retrieval is a weaker prior trigger than relational comparison.

E.2. Per-(domain, shape) prior-override table

Table 6 reports per-(domain, shape) prior-override counts for Sonnet on the extreme-flip panel (extreme) and subtle-flip panel (subtle) panels, across Table-only, Tools-curated, and Tools-open. Each cell is `pair_A_PO + pair_B_PO` out of 3 reps each (so 6 trials total per cell). Empty entries are `0/6`. This breakdown is what the main text's "three robustness axes" paragraph (Section 4.2) summarises.

Panel	Domain	Shape	Table-only PO	Tools-open PO	Tools-curated PO
extreme-flip panel (extreme)	D1 countries-pop	B	3+3	3+3	3+3
	D1 countries-pop	D	3+2	3+3	3+2
	D1 countries-pop	C	—	—	—
	D2-D5 (others)	B/D/C	—	—	—
subtle-flip panel (subtle)	D1 countries-pop	B	3+3	3+0	3+0
	D1 countries-pop	D	3+3	—	—
	D2 EU area	B	0+3	—	—
	D3 city GDP	B	0+3	0+3	0+3
	D3 city GDP	D	0+3	—	—
	D4 Forbes	B	3+0	—	—
	D5 Alpine peaks	B	0+3	—	—
	D5 Alpine peaks	D	0+3	—	—

Table 6. Per-(domain, shape) prior-override counts on Sonnet, Tier 1.5 panels extreme-flip panel + subtle-flip panel. Each cell is `pair_A + pair_B` out of 3 reps each. Domain abbreviations: D1 countries-population, D2 EU country area, D3 European city GDP, D4 Forbes top global companies, D5 Alpine peaks. Shapes: B binary-compare, D direction (larger/smaller), C cell-lookup or sum control. **Reading:** extreme-flip prior-override concentrates on countries-pop only (other 4 extreme-flip panel domains are 100% SF on Sonnet). subtle-flip prior-override generalises across 4 of 5 subtle-flip panel domains on Table-only; on Tools rungs (with tool access) it survives only on countries-pop and city-GDP (Madrid vs Berlin), the two strongest- prior-anchor pairs. EG-VAR is 100% source-faithful on the in-calculus shape-B and shape-C subset across both panels.

What the breakdown reveals. The narrowing → replication arc the main text claims is visible row-by-row. Under extreme-flip flips, all PO concentrates on D1 (countries-pop) regardless of rung. Under subtle-flip flips, Table-only PO spreads to 4 of 5 domains (D1, D2, D3, D5 trip; only D4 is clean on Table-only pair-B but trips on Table-only pair-A); but tools rescue all PO except on D1 (Egypt-Yemen) and D3 (Madrid-Berlin) — the two canonical strong-prior pairs. The same-tool baseline is therefore not just bounded but *specifically* bounded: it converges on the two pairs whose parametric prior is most entrenched.

E.3. Authority-cue ablation: no-cue pilot vs authority-cued pilot per-claim

Table 7 reports per-claim source-faithful (SF) / prior-override (PO) counts on the no-cue pilot countries-population pilot *without* an authority cue (no-cue pilot, 12 base claims) and *with* an authority cue (authority-cued pilot, "According to the table below, . . . " prefix on the same 12 claims). The strongest discriminator is V3_C1 (binary compare "which country has a larger population: Egypt or Yemen?"), which stays 0/3 SF on every Tools rung even with the cue. EG-VAR goes 3/3 SF on the same V3_C1 claim (see Section I for the trace).

Why V3_C1 is the canonical hard case. One natural response to a prior-override finding is "the model was not told to use the table." The authority-cued pilot ablation explicitly includes an "according to the table" anchoring cue in the prompt. Sonnet still selects the prior on V3_C1, 3/3 reps, on both Tools-curated (menu-enforced) and Tools-open (full tool surface). The kernel discipline closes this gap structurally — not because the LLM understands the cue better, but because the proof body `refine ⟨''yemen'', ...⟩` typechecks only when the attestation is `yemen`.

Claim	Shape	no-cue TC	no-cue TO	cued TC	cued TO	cued T-only
V3_C1	binary-compare	**0/3 (3 PO)**	**0/3 (3 PO)**	**0/3 (3 PO)**	**0/3 (3 PO)**	**0/3 (3 PO)**
V3_C2	boolean threshold	2/3 (1 PO)	1/3 (2 PO)	3/3 SF	2/3 (1 PO)	0/3 (3 PO)
V3_C3	negation compare	3/3 SF	3/3 SF	3/3 SF	3/3 SF	3/3 SF
V3_C4	rank position	3/3 SF	3/3 SF	3/3 SF	3/3 SF	3/3 SF
V3_C5	3-way compare	3/3 SF	3/3 SF	3/3 SF	3/3 SF	0/3 (3 PO)
V3_C6	threshold count	3/3 SF	3/3 SF	3/3 SF	3/3 SF	3/3 SF
V3_C7	categorical y/n	3/3 SF	3/3 SF	3/3 SF	3/3 SF	3/3 SF
V3_C8	ranking	3/3 SF	3/3 SF	3/3 SF	3/3 SF	3/3 SF
V3_C9	implicit compare	1/3 (2 PO)	0/3 (3 PO)	3/3 SF	3/3 SF	0/3 (3 PO)
V3_C10	diff vs literal	3/3 SF	3/3 SF	3/3 SF	3/3 SF	3/3 SF
V3_C11	cell lookup	3/3 SF	3/3 SF	3/3 SF	3/3 SF	3/3 SF
V3_C12	sum aggregation	3/3 SF	3/3 SF	3/3 SF	3/3 SF	3/3 SF
Totals		30/36 (83.3%)	28/36 (77.8%)	33/36 (91.7%)	32/36 (88.9%)	23/36 (63.9%)

Table 7. Per-claim source-faithfulness on the countries-population counterfactual pilot, comparing the no-cue baseline with the authority-cued version ("According to the table below, . . . " prefix). Columns: **TC** = Tools-curated, **TO** = Tools-open, **T-only** = Table-only. Sonnet 4.6, temperature 0, 3 reps per cell. The cue rescues V3_C2 (boolean threshold) and V3_C9 (implicit direction) on Tools rungs but does *not* rescue V3_C1 (binary compare Egypt vs. Yemen) on any rung; Table-only (no tools) is unaffected by the cue's claim-level intent on the binary-compare case (V3_C1 still 3 PO) and additionally regresses on V3_C2, V3_C5, and V3_C9 because mental-arithmetic-without-tools is prior-vulnerable on subtle compares. EG-VAR is 3/3 SF on V3_C1 (Section I) — the strongest possible discriminator.

E.4. Pre-registered success criteria for extreme-flip and subtle-flip panels

The Tier 1.5 extreme-flip panel (extreme flips, 5 domains $\times$ 2 pairs $\times$ 3 shapes = 30 claims) and subtle-flip panel (subtle flips, 5 stable-scalar-prior domains, same shape) were each designed with pre-registered success criteria recorded in `specs/tier1-v4-multidomain-panel.md` before any runs landed. We reproduce them here so the methodology can be audited against the result tables.

Pre-registered metric	Criterion
Tools-curated shape-B PO across 5 domains	$\geq 30\%$ of trials, on $\geq 3/5$ domains
Tools-curated shape-C control PO	$\leq 10\%$ of trials (lookup/aggregation stays SF)
EG-VAR shape-B + C source-faithful	$\geq 95\%$ across all domains
Bare-vs-anchored phrasing delta on shape B	if anchored $\rightarrow$ 0% PO on all 5 domains, "kernel discipline" framing becomes "anti-distractibility" instead
Tools-open $\geq$ Tools-curated in shape-B PO	tool surface is not protective

Outcomes against pre-registration. The extreme-flip panel *failed* the first criterion: Tools-curated shape-B PO concentrated on 1/5 domains (countries-pop) under extreme flips, not the 3/5 we pre-committed to. We reported this honestly in the main text ("narrowing" framing in Section 4.2). The failure motivated the subtle-flip panel redesign (subtle flips on stable-scalar priors), where the criterion lands: subtle-flip panel shape-B PO on Table-only reaches 4/5 domains; on Tools rungs it drops to 2 specific pairs (Egypt-Yemen + Madrid-Berlin) but the cross- domain replication is visible.

The shape-C control criterion (PO $\leq 10\%$) holds in both panels across all rungs.

The EG-VAR $\geq 95\%$ criterion is met (100% in every cell on the in-calculus shape-B + C subset).

The authority-cue criterion was *partially* met — anchored phrasing rescued V3_C2 + V3_C9 but not V3_C1; we reframed the finding as "cue rescues some shapes, not the strongest" rather than recategorising the discriminator (Section E.3).

The Tools-open $\geq$ Tools-curated criterion holds on both panels (Table 6).

F. Tier 1 data: fixture properties and 6-claim differential

F.1. The 6-claim differential: where EG-VAR wins on Tier 1

The Tier 1 ladder gap (EG-VAR 120/120 vs Tools-curated 114/120) is 6 claims at fixed tool surface. Table 8 reports exactly which 6 claims EG-VAR wins on, what Tools-curated outputs instead, and which failure mode each represents.

idx	Predicate	Claim (truncated)	Gold	Tools-curated answer (wrong)
12	PProd	Which time has the lowest local magnitude?	27 nov 1954	08:36 (time-of-day)
24	SumWhere	Total population of regions where Manchu % $>$ threshold	3,123,625,869	153,259,452 (subset only)
46	CountWhere	How many individuals have "Number" $>$ 1500?	14	40
52	CountWhere	How many parties have $>$ 10% of total votes?	3	4
67	PProd	Difference in goals between top-scoring forward and top-scoring midfielder	7	6
77	MeanWhere	Average US viewers (millions) for Season 1 episodes	8.45	6.29

Table 8. The 6 Tier 1 claims that distinguish EG-VAR from Tools-curated at fixed tool surface. EG-VAR outputs are kernel-checked Sigma witnesses matching the curator's gold; Tools-curated outputs are free-text answers that fail the answer-match judge.

Failure-mode taxonomy of the 6 wins. **Ambiguity (idx 12)** — "which time" admits two readings: the date or the time-of-day. The curator's typed goal commits to a specific reading (`HasProperty` e "`date`" v); Tools-curated picks the time-of-day reading and fails the answer-match. Kernel discipline + curator's commitment is what closes this gap, not proof checking alone — this is the "kernelized fixed-goal discipline at fixed tool surface" framing of the §4 result.

Composition arithmetic (idx 24, 46, 52, 67) — Tools-curated issues a partial tool sequence (one filter or one aggregate) and emits a free-text answer composed from incomplete evidence. EG-VAR emits the full PProd / SumWhere / CountWhere proof composing every required attestation; the kernel rejects partial proofs.

Subset confusion (idx 24) — Tools-curated aggregates over the wrong condition subset (the threshold filter is dropped on one of the two SumWhere components inside the PProd). The kernel-checked proof has both component types fixed by the curator's goal, so the solver cannot drop the filter without the proof failing.

Numeric format (idx 77) — Tools-curated reports "6.29" (off-by-one row included; computed mean over the wrong row set). EG-VAR's Sigma witness `Value.rat (mkRat 169 20) = 8.45` is the correctly-typed exact-rational answer.

These six claims — not the abstract architectural argument — are what the 5 pp Tier 1 gap actually *is*. Each is a place where the curator's typed goal makes the right answer derivable and the free-text baseline does not enforce that constraint.

F.2. Tier 1 fixture properties

We summarise the Tier 1 fixture composition:

Property	Value
Claims	120
Tables	120 (one claim per table)
PProd-composed claims	15 (12.5%)
PLift-using claims (arithmetic identity)	14 (11.7%)
Evidence count per claim (median / max)	1 / 4
Mean evidence count	1.1

L2 predicate distribution. The headline predicate covers the following Tier 1 claim shapes:

L2 predicate	Claims	%
`CountWhere`	32	26.7%
`MeanWhere`	32	26.7%
`SumWhere`	22	18.3%
`ArgmaxWhere`	19	15.8%
`MaxWhere`	9	7.5%
`ArgminWhere`	3	2.5%
`TopKSumByWhere`	1	0.8%
`MinWhere`	1	0.8%
`HasProperty`	1	0.8%

Coverage and limits. The fixture is biased toward single-shot aggregations (88% of claims have evidence-count = 1), with PProd composition tested explicitly on the 15 multi-evidence cases (12.5%). Tier 1 is constructed for breadth across L2 predicate shapes (9 distinct predicates) at relatively shallow composition depth. Deeper composition (PProd of 4+ evidence with `PLift` arithmetic identities, e.g., Section O) appears in 11.7% of claims. Tier 2 (Section 4.3) exercises the same fixture under LLM formalization; deeper composition remains a target for future work.

G. Reproducibility and formalization provenance

G.1. Reproduction artifacts

- Code supplement: https://anonymous.4open.science/r/anonymous-project-7B87/ (Lean 4 library, Python runtime, trust-artifact JSONs, 11 replay fixtures, runner scripts for E1–E4; see README).
- Reproduction scripts (Tier 1): `run_dg_eval.py`, `run_di_baseline.py`, `run_dj_no_tools.py`; Tier 1.5: `test_counterfactual.py`, `test_counterfactual_dg.py`, `test_counterfactual_dj.py`, all under `scripts/tier1/`.
- Logs written by the runners above: `results/tier1_{dg,di_strict,di_loose,dj}_*.jsonl` (E1), `results/cf_v{4,45}_*.tsv` (E2), `results/tier2_*.jsonl` (E3), `results/cf_v4_tier2_*.jsonl` (E4).
- Models: `claude-sonnet-4-6`, `claude-haiku-4-5-20251001`, and `claude-opus-4-7` (Tier 2 capability-scaling), temperature 0, identical prompt scaffolding across rungs.

Replay audit gate. Every generated Lean module at `results/generated/pipeline_<hash>.lean` can be replay-audited post-hoc by the standalone script `verify_axioms.py`. The script parses the `#print axioms claim_proof` output and checks the dependency closure against the whitelist buckets used in our analysis: (i) the fixed EG-VAR prelude, (ii) Lean 4 standard core axioms (`Classical.choice`, `Quot.sound`, `propext`), (iii) runtime-emitted observation-leaf schemas matching `obsN_*` or `{table,stats}Tool` (full emission-site table in Table 9), and (iv) the per-source lift prefix `lift_<sourceHash>_*`. The trust-artifact discipline requires either `mkVerified` (attestation path) or at least one runtime leaf together with at least one per-source lift (the direct-evidence lift path per Section 3.2). Any axiom outside these buckets, or any `sorry`/`sorryAx` in the closure, fails the audit. The primary safety gate at generation time is the Lean kernel call inside `pipeline.py` (through `lake env lean`); the replay audit is an additional artifact-level check that any third party holding the generated module bundle and the Lean toolchain can re-run via `python -m runtime.verify_axioms`, without invoking any LLM or re-running the analyst/formalizer/solver roles.

G.2. Formalization provenance

This subsection expands the main-text summary at Section 3.6. We document the curation pipelines explicitly because any reader of Theorem 3.1 must distinguish runtime-minted trust artifacts from pre-committed ones.

Tier 1 and Tier 1.5 fixtures (`data/tier1_gold_formalizations.json`, `data/tier1_table_formalizations.json`)**.** These fixtures supply the pipeline-produced, fixture-committed goal type τ^{gold} and the per-source audited formalization $\Lambda(s)$ for every claim in the four-rung ladder (Sections 4.1 and 4.2). Each entry was produced by an automated subagent pipeline frozen per git commit:

1. **Proposer** (Claude Sonnet subagent, temperature 0, frozen prompt): drafts a candidate Lean goal type and per-table menu, reading only the raw TableBench row data and column headers.
2. **Critic** (Codex GPT-5.4 subagent, separate session per batch): reviews for anti-cheating violations, shape correctness relative to the claim, and R10 menu soundness per Section 3.5.
3. **Kernel typecheck** (`lake env lean`): gates inclusion; any entry whose declared type fails the Lean elaborator is rejected and returned to the proposer.
4. **Human meta-supervision**: a single human (the author) reviewed batch-level acceptance rates and systemic failure categories (e.g., "pipeline emits raw `<<col>>` while lift canonicalizes" as a structural bug), but did *not* edit any individual formalization. No per-entry manual fixes; all fixture content descends from the proposer–critic–kernel loop.

At fixture close the two JSON files are frozen under the commit hash reported in Section G. Curator yield (entries that survived stages 1–3) was $120/120$ for Tier 1 and $30/30$ per panel for Tier 1.5, covering shape-B (binary-compare), shape-C (cell-lookup), and shape-D (categorical-direction) claims. Shape-D claims reuse the shape-B formalization plus a deterministic post-processor; no kernel extension is required.

Working dictionary (formalizer state). The formalizer is memoryless across claims but has access to a *working dictionary* of Property, Entity, and Convention entries accumulated from prior formalizations and from the analyst pass on the current source. Dictionary entries record canonical column names, value formats, and known encoding conventions. The dictionary is a curator-controlled cross-claim memory; it does not give the formalizer access to the gold answer or solver-side information.

Tier 2 evaluation. The Tier 2 runtime passes the natural-language claim to an LLM formalizer at generation time, with access to the same per-source menu $\Lambda(s)$ (as a trust artifact, not as feedback) but no gold goal type. The formalizer proposes its own τ^{LLM}; the solver then attempts a proof against that type; the kernel accepts only whitelist- compliant proofs. Section 4.3 reports the resulting semantic-faithfulness decomposition on $n{=}120$ TableBench claims.

What $\Lambda(s)$ is and what it is not. The per-source menu $\Lambda(s)$ is the curator's interpretive commitment that a given tool's L1 output (e.g., `CountCells`) lifts to a specific L2 world-property predicate (e.g., `CountWhere`) on this source. It is *not* a semantic certificate: a wrong but audited lift can still certify a wrong formalized claim, as we document in Section 5.1(ii). The provenance chain terminates at the proposer–critic–kernel loop above; the audit at that loop is the trust artifact. We use the term "audited per-source formalization" throughout to keep the term separable from semantic-faithfulness claims.

What prevents formalizer collusion. At evaluation time the Tier 1/1.5 runtime loads τ^{gold} from the frozen fixture and never invokes the formalizer role. The kernel-trusted prelude, $\Lambda(s)$, and the runtime-minted observation leaves are the only axiom sources (Section 3.6); the solver writes only the tactic body. Consequently, no information from the gold answer or curator's adjudicated reading enters the claim proof at runtime — only pre-committed artifacts do.

H. Worked Tier 1 examples: end-to-end trace

H.1. HIV argmax via state projection

Suppose the claim is *"Which state has the highest HIV incidence rate?"*. EG-VAR proceeds: (1) The solver requests the per-source operation `argmax_with_id` on column "incidence rate". (2) The runtime executes the operation and returns an attested payload `Attested_T q p` that contains the maximum value v, the row handle r at which it occurs, and a cell projection `CellAt` r "state" "alabama". (3) The kernel's interpretation function lifts the payload to two L2 facts: `MaxCell ``incidence''` $c$ $r$ $v$ and `CellAt` r ```state'' ``alabama''`. (4) The per-source lift composes them:

```
axiom hiv_lift_argmax_via_state_proj
  (col : String) (c : Cond) (r : RowHandle)
  (name : String) (v : Value) :
    Evidence Verified (MaxCell col c r v) ->
    Evidence Verified
      (CellAt r "state" (Value.str name)) ->
    Evidence Verified
      (ArgmaxWhere col c name v)
```

(5) The solver writes a refine tactic with witness `"alabama"` and applies the per-source lift; the kernel mints an `Evidence Verified` witness for `ArgmaxWhere "incidence"` c `"alabama"` v, and the Sigma witness "alabama" is the certified answer.

Identity in EG-VAR rides on names rather than category-kinds: "alabama" is a `String`, not an inhabitant of an opaque `State` type. This is the nominalist choice (Quine, 1969; Burgess & Rosen, 1997) — identity-bearers are the names the table actually emits, so audit proceeds by comparing strings rather than inferring category membership. The kernel never has to commit to an ontology of states; the curator commits to a column whose values *are* the identifiers, and the lift makes that commitment typed and inspectable.

A per-source lift is *table-specific* (it encodes the curator's commitment that "state" is the row-identifying column for this

particular table) but *claim-generic* (parametric over measure column, condition, entity name, and value, so a single lift covers many argmax claims). Lifts are audited once per source then trusted kernel-side for all subsequent claims over that source. Absence of a matching lift is a feature: the solver has no L2 proof path, the kernel rejects, and the system abstains rather than silently guessing. The guarantee is conditional: a semantically *wrong* audited lift can still produce a verified output for a formalized claim that the lift's semantics admit. EG-VAR rules out unsupported claims between tool and output; it does not eliminate curator error in the trust artifact itself.

This is the concrete protocol behind our claim that the formalization barrier in empirical reasoning is a curation gap, not an expressive limit (Section 1). The L1/L2 separation is portable: knowledge graphs, SQL views, JSON APIs, and sensor streams admit the same per-source-lift pattern. We sketch the generalization in Section 5.2.

H.2. Full Tier 1 trace

We walk through a single Tier 1 claim from natural-language input to kernel-checked Sigma witness, to make the architecture's per-step trust commitments concrete.

Claim and goal type. *"Which crater has the largest diameter?"* over the lunar-crater table. The fixture-committed Lean goal (in Lean's dependent-sum notation):

```
Sigma (e : String), Sigma (v : Value),
  Evidence Verified
    (ArgmaxWhere c_diameter
                 Cond.all e v)
```

where `c_diameter` = `"diameter (km)"`. The Sigma binders make the certified answer the witness e (Martin-Löf, 1984); the surface answer is the first projection.

Step 1 (solver → runtime). The solver issues the structured action {`action: ``stats'', operation: ``argmax_with_id'', column: ``diameter (km)'', identity_column: ``name''`}.

Step 2 (runtime → attested payload). The runtime executes the operation deterministically against the table; it returns an `Attested_T q p` witness with payload {MAXROW: `cleopatra`, MAXVALUE: VALUE.RAT (MKRAT 78 1)}.

Step 3 (interp_T → L1 facts). The interpretation function for `argmax_with_id` reads the attested payload into two typed L1 storage facts: `MaxCell ``diameter (km)'' Cond.all 78` (the maximum value at the storage layer) and `CellAt` r ```name'' (Value.str ``cleopatra'')` (the identity-column value at row r).

Step 4 (per-source lift → L2 claim). The lunar-crater `argmax_via_name_proj` lift composes the two L1 facts and emits the L2 claim `Evidence Verified (ArgmaxWhere ``diameter (km)'' Cond.all ``cleopatra'' 78)`. The lift is the curator's interpretive commitment that "name" is the row-identifying column for this table.

Step 5 (solver → proof term). The solver writes `refine ⟨``cleopatra'', ..., ?_⟩; exact argmax_via_name_proj ...`; the kernel type-checks and the Sigma witness "cleopatra" is extracted as the certified answer.

The generated Lean file is the replay artifact (G1 in Section 5.2); the audit cost is the per-source lift audit, paid once and reused.

I. Failure cases and formalizer-design notes

I.1. V3_C1 (Egypt-vs-Yemen): baseline prior-override

Setup. Synthetic counterfactual countries-population table: Egypt's `july 1, 2013 projection` cell overwritten to `12` (real ∼85M). Yemen unchanged at ∼25M. Question: "Which country has a larger population: Egypt or Yemen?".

Sonnet Tools-curated trace (3/3 prior-override). The model issues a `cell_lookup` for Egypt, receives `12`; issues a `cell_lookup` for Yemen, receives `25252000`; then answers "Egypt". The tool returns the source values; the model overrides them. The same pattern occurs across all three reps.

EG-VAR (3/3 source-faithful). The kernel mints an `Evidence Verified` witness from the per-source argmax-with-disjunction lift on the attested payload. The solver writes `refine ⟨‘‘yemen’’, ...⟩; exact ....` The Sigma witness "yemen" is the certified answer.

Authority-cue ablation (authority-cued pilot). Prefixing the question with "According to the table below, ..." rescues V3_C2 (boolean threshold) and V3_C9 (implicit direction) on Tools-curated but does not rescue V3_C1 (still 3/3 prior-override). The cue is not strong enough to override the parametric prior on famous-entity populations; the kernel attestation is.

I.2. Formalizer failures: runtime-vs-fixture delta (Tier 2)

Section 4.3 reports the end-to-end Tier 2 decomposition at n=120; this appendix zooms in on the five Sonnet cases where the runtime formalization diverged materially from the fixture-standard goal type. Four produce kernel-accepted but answer-divergent output (B5b); one aborts the formalizer pass (B7). We present each as a structured delta, organised by the compositional axis along which the divergence occurred. Cases classified B5a (recorded ambiguity) and B5c (benchmark-artifact issue) are excluded here; they are presented case-by-case in the main-text Table 5 caption and do not represent formalizer-attributable error.

Across these five cases, the common pattern is that the formalizer makes a local-reading decision at a composition node (which `Cond` constructor, which column, whether to include a `TopK` constraint) that a curator with the claim and the fixture-standard formalization in front of them would have taken differently. Each case is addressable by supervised fine-tuning on fixture (claim, gold `lean_goal_type`) pairs or RL with a gold-answer reward; we discuss the direction at the end of the section.

I.3. Composite-column matching strategy (idx 50)

Claim. "How many hurricanes occurred in the month of September?"

Delta. Column `month` contains date-string values of the form `‘‘September 1979’’`, `‘‘October 1988’’`, etc. (month-plus-year strings). Fixture uses `Cond.propContains`:

```
CountWhere "month"
  (Cond.propContains "month" (Value.str "September"))
  ans
```

Runtime formalizer chose `Cond.propEq`:

```
CountWhere "month"
  (Cond.propEq "month" (Value.str "September"))
  ans
```

Runtime answer: 9. Fixture answer: 19. The formalizer's `ambiguous_terms` field recorded the choice explicitly as "*Cond.propEq — exact match*". The kernel accepted the proof under the LLM's chosen reading; the L1 tool correctly returned `count = 9` for strict equality with the literal string `‘‘September’’`, which matches zero `month = "September ..."` rows in the table.

Character of the error. The formalizer recognised the existence of a matching-strategy decision, logged it, and picked the strategy whose answer would be correct *only if* the `month` column's values were bare month names. It did not use the column's actual value format (exposed by the analyst's reading guide) to discriminate.

I.4. Dropped `TopK` constraint (idx 61)

Claim. "What is the average market value of the top 5 companies in the oil and gas industry?"

Delta. Fixture uses `TopKMeanWhere`, which the per-source menu includes for this column:

```
TopKMeanWhere "market value (billion )" 5
  (Cond.propEq "industry" (Value.str "oil and gas"))
  ans
```

Runtime formalizer dropped the top-5 constraint and used plain `MeanWhere`:

```
MeanWhere "market value (billion )"
  (Cond.propEq "industry" (Value.str "oil and gas"))
  ans
```

Runtime answer: $1208/5$. Fixture answer: 276.06. The formalizer's `ambiguous_terms` names the phrase "*top 5 companies in the oil and gas industry*" and records the chosen formalization as `MeanWhere`, so the dropped `TopK` was deliberate, not an oversight.

Character of the error. The formalizer treated "top 5" as rhetorical rather than compositional. The per-source menu includes `TopKMeanWhere` for this column; the formalizer's compositional-operator selection step did not consult it.

I.5. Column disambiguation (idx 67)

Claim. "What is the difference in total goals scored by the top-scoring forward (fw) and the top-scoring midfielder (mf) in the league?"

Delta. Table has both a `league goals` column and a `total goals` column. Claim literally says "total goals". Fixture uses `total goals`:

```
MaxWhere "total goals" (Cond.propEq "position" "fw") a
... MaxWhere "total goals" (Cond.propEq "position" "mf") b
... (PLift (ans = a - b))
```

Runtime formalizer picked `league goals`:

```
MaxWhere "league goals" (Cond.propEq "position" "fw") a
... MaxWhere "league goals" (Cond.propEq "position" "mf") b
```

Runtime answer: 6. Fixture answer: 7.

Character of the error. Column-selection under near-synonym ambiguity, where the claim's exact wording ("*total* goals") is the dispositive signal. The formalizer's `properties_needed` did not enumerate a candidate-set check against the claim's noun phrase.

I.6. Condition-conjunct erasure and self-inconsistency (idx 94)

Claim. "What is the average prominence of mountain peaks in the Democratic Republic of the Congo that have an elevation of at least 3000 meters?"

Delta. Fixture:

```
MeanWhere "prominence (m)"
  (Cond.and
     (Cond.propContains "country"
       (Value.str "democratic republic of the congo"))
     (Cond.propGte "elevation (m)"
       (Value.rat (mkRat 3000 1))))
  ans
```

Runtime formalizer:

```
MeanWhere "prominence (m)"
  (Cond.and
     (Cond.propGte "elevation (m)"
       (Value.rat (mkRat 3000 1)))
     (Cond.propGte "col (m)"
       (Value.rat (mkRat 0 1))))
  ans
```

The country filter is absent; the second conjunct is `col (m)` $\geq$ `0` (trivially true on this table). The formalizer's `ambiguous_terms` field claims the chosen filter is `Cond.propEq "country" "democratic republic of the congo"` — *but this constructor does not appear in the emitted* `lean_goal_type`. Runtime answer: $14075/6$; fixture answer: 2606.25.

Character of the error. Self-inconsistency between the formalizer's self-reported decision (`ambiguous_terms`) and the actual `lean_goal_type` it emitted. A post-formalizer consistency pass — verifying every adjudication recorded in `ambiguous_terms` is reflected in the emitted goal — would catch this class mechanically at audit time. We list this as a near-term infrastructure improvement in Section 5.3.

I.7. Compositional-complexity abort (idx 34)

Claim. "What is the total number of medals (M36 + M36B1 + M36B2) earned from May 1944 to August 1944?"

Delta. Fixture is a 4-level dependent sum:

```
Sigma ans, Sigma a, Sigma b, Sigma c,
  PProd (SumWhere "m36"   <4-way month or> a)
  (PProd (SumWhere "m36b1" <same or> b)
  (PProd (SumWhere "m36b2" <same or> c)
  (PLift (ans.asRat = a.asRat + b.asRat + c.asRat))))
```

The fixture formalization composes *three separate measure columns* with *a 4-way disjunction over the month column* and *a 3-term arithmetic* combining the resulting witnesses. Runtime formalizer emitted an empty `lean_goal_type` and aborted on its first attempt; the pipeline returned ABSTAIN rather than a wrong formalization.

Character of the error. Compositional complexity boundary: the claim requires simultaneous choices across three dimensions (measure column, filter disjunction width, arithmetic composition shape) in a single pass. The formalizer's single-pass architecture is not expressive enough to emit this class of goal reliably. Note that this is the *desirable* failure mode: no silent-wrong, no Verified output with the wrong answer. The system abstained honestly; the downstream consumer sees ABSTAIN and can route elsewhere.

We note that the *decomposer + binder* two-stage formalizer pattern — splitting the goal into per-measure sub-claims and composing them via a binder role — is a natural architectural response to this class; we describe it in Section 5.3 as future work.

I.8. Taxonomy and improvement direction

The five cases partition into five orthogonal compositional axes of the formalizer's output: matching strategy (idx 50), operator selection (idx 61), column selection (idx 67), condition-conjunct preservation (idx 94), and compositional arity (idx 34). Three of these (idx 50, 61, 67) are *local* decisions: each resolves to a choice among a small candidate set that is determined by the claim's wording together with local information already available to the formalizer (column-value format for idx 50; per-source menu for idx 61; claim noun phrase for idx 67). Case idx 94 is a *consistency* gap between two of the formalizer's own outputs. Case idx 34 is a *capacity* limit rather than a reading error.

Near-term infrastructure improvements. The idx 94 self-inconsistency case is mechanically detectable: verifying agreement between each entry of `ambiguous_terms.chosen` and the emitted `lean_goal_type` is a deterministic post-formalizer audit that would convert this failure mode into an ABSTAIN or a retry. We list this as future work rather than a claimed feature.

RL fine-tuning of the formalizer. Each of the four reading-error cases (idx 50, 61, 67, 94) is a single-claim adjudication where the fixture's `lean_goal_type` supplies a ground-truth choice at exactly the compositional node where the runtime formalizer diverged. The fixture, documented with provenance in Section 3.6, therefore supplies a structured supervised-fine-tuning corpus of (claim, analyst-guide, per-source menu, gold `lean_goal_type`) tuples. A natural next step is to fine-tune a formalizer model on this corpus, either supervised or with a verifier-based RL loop where the reward is kernel acceptance against the fixture goal type. The verifier is already on hand: it is the same kernel the paper uses to certify production

outputs. This produces a formalizer whose local reading decisions converge to the curator's, without weakening any of the runtime trust properties established in Section 3.6.

Why this is compatible with the paper's safety claims. Fine-tuning the formalizer is an *accuracy* intervention, not a *safety* intervention: Theorem 3.1 and Theorem 3.2 are properties of the runtime + kernel, invariant under any formalizer. A fine-tuned formalizer changes the VERIFIED/ABSTAIN split favorably without changing the no-silent-wrong guarantee.

J. Abstention case study

Setup. A Tier 1 claim whose goal type required a per-source `topk_sum_by` lift over a measure column not registered in the curator's lift menu for that table.

EG-VAR outcome. The kernel rejects every proof attempt: no L2 fact unifies with the goal vocabulary because the lift menu has no matching entry. The system reports ABSTAIN with the unsatisfied goal type.

Tools baseline outcome. The same-tool baseline calls a related stats operation, receives a numeric value, and emits a free-text answer that happens to match the gold (a partial correctness coincidence; the operation called was not the one the gold formalization required). Verified-wrong vs verified-right is indistinguishable downstream without a kernel-checked proof.

Governance reading. EG-VAR's abstention is machine-actionable (G3 in Section 5.2); the baseline's "answer" is opaque. The trade-off is honest: EG-VAR loses one match-counted claim, but the failure mode is structurally typed.

K. Per-source formalization: theory anchors

K.1. Robustness hierarchy

The argument that records-vs-entities is hierarchical, not categorical (Section 3.2), can be made precise. Storage features that hold prior to any interpretive commitment: same-row entries share a row-anchor, same-column entries share a column-anchor. Layering interpretive commitments on these invariants yields a four-level hierarchy of per-source formalizations:

Level	Commitment	L2 vocabulary unlocked
1. Aggregate-only	none beyond layout	`SumWhere`, `CountWhere`, `MeanWhere`, `MaxWhere`, `MinWhere`
2. Row-cell facts	rows have stable handles	row-anchored `cell_lookup`
3. Entity-anchored	rows are entities; col X gives names	`HasProperty`, `ArgmaxWhere`, `EntitySum`
4. Relational	both axes entity-identified	`Relation`, matchup-style

Each level's commitments include those above. A record table stops at level 1; an entity table reaches level 3; a matchup table reaches level 4. Same kernel, same L1/L2 split, same ontology — different stopping points on a single hierarchy. "Fail-closed for mismatched claims" becomes a hierarchy property: a claim requiring level-3 commitments against a level-1 table has no L2 fact to unify against, so the pipeline abstains.

K.2. Rank as interpretation: gauge freedom

A given dataset admits multiple consistent rank interpretations. A matchup table can be formalized as $\mathsf{Result} : \mathsf{Player} \to \mathsf{Player} \to \mathsf{Outcome}$ (rank-2 curried), $\mathsf{Result}' : \mathsf{Player} \times \mathsf{Player} \to \mathsf{Outcome}$ (rank-2 uncurried), or $\mathsf{MatchOutcome} : \mathsf{Match} \to \mathsf{Outcome}$ where $\mathsf{Match} := \mathsf{Player} \times \mathsf{Player}$ (rank-1 with composite domain). All three encode the same function and are type-isomorphic in any Cartesian closed setting (Mac Lane, 1971).

Rank-shifting interpretations across the lambda calculus reduce to currying/uncurrying, which Lean 4 makes definitionally transparent ($A \to B \to C$ is sugar for $A \to (B \to C)$). The kernel does not need to commit to "the true rank" of a table; it accepts any consistent per-source lift catalogue. Other examples of the same ambiguity:

- *Time series.* Rank-1 (time $\to$ value) vs rank-2 with channels (time $\times$ channel $\to$ value).
- *Pixel grid.* Rank-2 (row $\times$ col $\to$ intensity) vs rank-3 (row $\times$ col $\times$ channel $\to$ value).

- *RDF triples.* Rank-1 over triples; rank-2 (subject, predicate) → object; rank-3 (subject, predicate, object) → {true, false}.

- *Relational tables with primary key.* Rank-1 over rows-as- records vs rank-2 (row, column) → cell.

These are not subjective conventions but type isomorphisms. The curator's job is to pick an interpretation that fits the anticipated claims and audit the lift menu under it; the kernel's job is to accept any consistent commitment.

K.3. The scaling argument: formalization gap shrinks as sources become formal

The empirical results in Section 4 are bounded by the formalization burden of CSV-style sources. As data substrates evolve toward typed interfaces, the formalizer's job shrinks while the kernel guarantee stays the same:

Substrate	Formalization burden
CSV	Column names are strings; values are free text. Maximal lift authoring per source.
SQL view	Schema is typed, values structured. Lift catalog auto-generates from view definitions.
OpenAPI	Typed inputs/outputs in spec. Lift = adapter from spec types to L2 ontology.
Knowledge graph	Entities + relations already typed. Lift = direct projection from RDF to L2.
Lean / formal source	Zero gap; compose proofs directly.

In the limit of fully formal sources, the LLM operates entirely within the formal world, the kernel guarantee is vacuous (because the substrate is already typed), and EG-VAR collapses to LLM-driven theorem proving with attested premises. The architecture is designed to interpolate: the same kernel + lift discipline runs across the substrate spectrum. We expect the kernel-vs-baseline gap on real-world deployments to narrow over time as data sources adopt structured interfaces, not because models improve, but because the formalization burden moves to the source curator and out of the inference path.

L. Tier 2 reporting taxonomy and capability-scaling check

This appendix expands the main-text table caption at Table 5 and reports the Opus capability-scaling check.

Category definitions. Categories are mutually exclusive and sum to 120. **B1**: kernel-accepted, type matches gold, answer matches gold. **B4**: kernel-accepted, type differs from gold but answer matches. **B5a** (*ambiguous claim, logged*): kernel-accepted divergent answer under a defensible alternative reading that the formalizer logged in `ambiguous_terms`. **B5b** (*semantic formalizer error*): pure formalizer error (wrong column, dropped constraint, composite-value matching-strategy error) — kernel still accepts the proof, but of a mis-formalized goal. **B5c** (*benchmark gold error, surfaced*): audited benchmark-artifact case (source-table HTML leakage, or arithmetic error in benchmark gold) that the system exposes rather than propagates; see Section I.2. **B6**: kernel-rejected, honest abstention. **B3**: solver exhausted tactic retries with type matching gold. **B7**: formalizer aborted.

Reading the decomposition. The key Tier 2 number is the pure end-to-end formalizer-error rate, not the flat disagreement rate. On Sonnet, only 4/120 cases are bona fide formalizer mistakes (B5b). The remaining divergences are *auditable*: 5 logged alternative readings (B5a), 2 surfaced benchmark-artifact issues (B5c), 3 honest abstentions (B6), and 5 solver or formalizer ceilings (B3+B7). Unlike B5b, B5a and B5c are explicit, inspectable disagreements rather than silent system error: each B5a entry's alternative-reading choice is logged in `ambiguous_terms` by the formalizer, and each B5c entry is an audited case study in Section I.2. Treating logged ambiguities separately raises auditable agreement to 88.3%; adding surfaced benchmark-artifact issues yields 90.0%.

Capability-scaling check. As a robustness check on the claim that residual formalizer error is capability-limited, we evaluate the stronger Opus model on the 9 Sonnet residual indices (B3/B7 or B5b). Opus converts four B3/B7 cases to accepted answer-matches and one B5b case to honest abstention; two B5b cases persist. The formalizer-error rate on this residual subset halves from 4/9 to 2/9; the theorem-level safety invariant is unchanged under either formalizer.

M. Scope limits and future extensions

This appendix expands the main-text summaries at Sections 5.1 and 5.3.

M.1. Limitations

(i) Formalizer scope. Tier 1 and Tier 1.5 bypass the NL-to-Lean formalizer step by loading fixture-committed gold typed goals (Section 3.6); they evaluate the safety and source-faithfulness properties under ideal formalization. Tier 2 (Section 4.3) removes that bypass and evaluates the LLM formalizer end-to-end. The 3-blind-formalizer consensus design is implemented in the runtime but not evaluated here; we defer the consensus study to future work.

(ii) Trust artifacts. Tools, per-source lifts, and the L1 adapter are trusted (auditable but not type-checked end-to-end). A semantically wrong audited lift can certify a wrong formalized claim; EG-VAR rules out unsupported claims between tool and output, not curator error in the trust artifact itself (Section 3.5).

(iii) Single-source scope. All evaluations are over single TableBench tables. Multi-source conflict, retrieval over distributed KGs, and time-varying sources are out of scope; we list near-term extensions below.

(v) World inconsistency containment. The implementation realizes Theorem 3.1 and Theorem 3.2 structurally; the companion world-inconsistency-containment property relies on the `WProp` $\not\to$ `Prop` phase separation in the current Lean encoding; we treat it as implementation detail, not a main-text theorem.

M.2. Future work scoped to governance reach

Fine-tuning the NL formalizer. Tier 2 (Section 4.3) reports residual semantic formalizer error at 3.3% (Sonnet) / 1.7% (Opus). Each of the 4 Sonnet B5b cases is a single-claim adjudication where the fixture supplies a ground-truth choice at exactly the compositional node where the runtime formalizer diverged, and the curator pipeline (Section 3.6) already provides a structured SFT / RL corpus. Fine-tuning a formalizer on this corpus, with kernel acceptance as the reward signal, is an accuracy intervention that tightens the residual rate without weakening Theorem 3.1; the verifier is already on hand — it is the same kernel the paper uses to certify production outputs. A 3-blind-formalizer consensus gate is similarly implemented but not evaluated here; we expect it to convert a further fraction of silent B5b mistakes into honest B6 abstentions.

Conflicting sources. Multi-source attestation introduces conflict between attested storage facts. The grade calculus admits explicit conflict-downgrade rules (VERIFIED $\to$ SUPPORTED on detected disagreement); empirical evaluation is future work.

Cross-substrate panel. Replicating the source-faithfulness discriminator on KG-grounded and SQL-grounded claims would directly test the L1/L2 portability claim (Section 5.2).

N. Theoretical anchors deferred from the main text

The main text uses a selected set of theoretical anchors as Justification Anchors and Lineage Paragraphs (Martin-Löf, Curry–Howard, Codd, Mac Lane, Quine, Burgess–Rosen, Raggi, Bundy, Clarke, Lahiri, Artemov, Fine, Green). The following anchors connect to the same design choices but are deferred here for space:

Source semantics and naming. Cooper (2012) (Type Theory with Records, TTR) gives a type-theoretic substrate for situated semantics that resembles our per-source-formalization commitment. Montague (1973) established that the natural-language $\to$ formal-meaning gap admits a typed denotational treatment, the genealogy of our claim formalization. Barwise & Perry (1983) (situation semantics) treats situations as first-class typed objects, the same move as our per-source formalization. Earlier in the lineage, Frege (1892) distinguished sense and reference — a distinction EG-VAR's name-as-string discipline collapses by choosing reference fixed at the storage layer. Following Kripke (1980) on rigid designators, our choice of name- columns as identity-bearers is the operational analogue of name- rigidity at the trust boundary. Donnellan (1966) distinguishes referential and attributive uses of definite descriptions; EG-VAR's lift menu enforces the referential reading ("the row whose name-column is X").

Uncertainty and conflict. Future multi-source evaluation opens connections to Belnap (1977) four-valued paraconsistent logic, Anderson & Belnap (1975) relevance/relevant logic, and Jøsang (2016) subjective-logic continuous-confidence.

Caminada & Amgoud (2007) ASPIC+ structured argumentation gives a substrate for grade-aware retraction under contradiction. Doyle (1979) JTMS truth-maintenance systems are the classical dependency-graph substrate for provenance-tracked retraction. Alchourrón et al. (1985) AGM belief revision frames the multi-source contradiction problem. EG-VAR's grade calculus accommodates these via the conflict-downgrade rules sketched in Section 5.3.

Optional / cautious anchors. Univalence in homotopy type theory (The Univalent Foundations Program, 2013) would give a principled account of equivalence classes of source formalizations (different rank readings, different lift catalogs); we mention this as a future- work analogy rather than a current design basis. Reliabilist epistemology (Goldman, 1979) grounds our "verified by reliable process" framing of VERIFIED evidence. Attribute-based access control (Hu et al., 2014) and typed semistructured data (Abiteboul et al., 2000) are the systems analogues of EG-VAR's per-source-formalization protocol.

Soundness over internal completeness. The kernel does *not* relate semantically equivalent L2 predicates by general lemmas: there is no built-in $\mathsf{MeanWhere}\ c\ v \Leftrightarrow \exists s, n.\ \mathsf{SumWhere}\ c\ s \wedge \mathsf{CountWhere}\ c\ n \wedge v = s/n$. Equivalences are localized in the per-source tools and lifts, where they are computed directly and audited offline; the kernel's proof search stays decidable and its proof traces stay small enough to inspect. This is the same fragmentation move that gives us decidable fragments of first-order logic, theory combination in SMT, and description logics: trade FOL completeness for tractable, auditable per-theory reasoning. Raggi et al. (2016) formalize precisely this: representation matters in proof, and semantically equivalent encodings can carry very different proof-engineering costs.

O. Worked Tier 1 examples: cross-tribunal PProd composition

The lunar-craters argmax in Section H exercises a single tool call and a single per-source lift. We now walk through a substantially more complex Tier 1 claim: a four-tool, four-evidence, PProd-composed claim with an arithmetic identity between the witnesses. This is representative of the multi-shot composition shapes we curated in Tier 1's PProd cluster.

Source. TableBench table `4fbaad0b...` (Spanish-Inquisition-tribunal autos da fé records). Relevant excerpt:

Tribunal	Autos da fé	Exec. persona	Exec. effigie	Penanced
Barcelona	8	1	1	15
Valencia	4	2	0	49
Córdoba	13	17	19	125
Cuenca	7	7	10	35
...				

Claim. *"How much greater is the total number of executions (in persona and in effigie) in Córdoba compared to Valencia?"* The arithmetic answer is $(17 + 19) - (2 + 0) = 34$.

Per-source tool menu. For this table the curator registered, among others:

- `stats_sum_strict` on column "executions in persona"
- `stats_sum_strict` on column "executions in effigie"
- Each accepts a `Cond.propEq` on the "tribunal" identity column.

The tool menu is the audit surface; a claim that requires a predicate not in the menu fails closed.

Lean goal type (fixture-committed).

```
Sigma (ans : Value),
Sigma (a1 : Value), Sigma (a2 : Value),
Sigma (b1 : Value), Sigma (b2 : Value),
PProd
  (Evidence Verified
     (SumWhere "executions in persona"
        (Cond.propEq "tribunal"
```

```
                    (Value.str "cordoba")) a1))
(PProd
  (Evidence Verified
     (SumWhere "executions in effigie"
        (Cond.propEq "tribunal"
                    (Value.str "cordoba")) a2))
(PProd
  (Evidence Verified
     (SumWhere "executions in persona"
        (Cond.propEq "tribunal"
                    (Value.str "valencia")) b1))
(PProd
  (Evidence Verified
     (SumWhere "executions in effigie"
        (Cond.propEq "tribunal"
                    (Value.str "valencia")) b2))
  (PLift (ans.asRat =
           (a1.asRat + a2.asRat) -
           (b1.asRat + b2.asRat))))))
```

The five Sigma binders fix: the surface answer `ans`, plus four numeric witnesses for the four sum sub-claims. The final `PLift` is the kernel-checked arithmetic identity that ties the witnesses to `ans`.

Solver action sequence (4 tool calls).

```
{ "action": "stats", "operation": "sum",
  "column": "Executions in persona",
  "condition": { "op": "eq",
    "property": "Tribunal", "value": "cordoba" } }
{ "action": "stats", "operation": "sum",
  "column": "Executions in effigie",
  "condition": { "op": "eq",
    "property": "Tribunal", "value": "cordoba" } }
{ "action": "stats", "operation": "sum",
  "column": "Executions in persona",
  "condition": { "op": "eq",
    "property": "Tribunal", "value": "valencia" } }
{ "action": "stats", "operation": "sum",
  "column": "Executions in effigie",
  "condition": { "op": "eq",
    "property": "Tribunal", "value": "valencia" } }
```

Tool attestations. The runtime returns `Attested_T` payloads: $a_1 = 17$ (Córdoba persona, single-row sum), $a_2 = 19$ (Córdoba effigie), $b_1 = 2$ (Valencia persona), $b_2 = 0$ (Valencia effigie).

L2 evidence via per-source lifts. Each attested payload is lifted via the `stats_sum_strict` per-source lift, which takes the L1 column-strict sum attestation and the `Cond.propEq` discriminator and produces an L2 `SumWhere` witness. The four lift applications produce `obs0_l2`, `obs1_l2`, `obs2_l2`, `obs3_l2`, each typed as `Evidence Verified (SumWhere ...)` with the right column/condition/value triple.

Solver proof body.

```
refine ⟨Value.rat (mkRat 34 1),
        obs0_val, obs1_val,
        obs2_val, obs3_val, ?_⟩
refine PProd.mk obs0_l2
       (PProd.mk obs1_l2
       (PProd.mk obs2_l2
       (PProd.mk obs3_l2
       (PLift.up ?_))))
native_decide
```

The first `refine` fills the five Sigma binders: the surface answer is `Value.rat (mkRat 34 1)` ($= 34/1$), and the four numeric witnesses are the values `obs0_val` ... `obs3_val` that the kernel already bound when the lifts fired. The second `refine` threads the four `Evidence Verified` witnesses through the right-associated `PProd` chain. `native_decide` closes the residual `PLift` goal: the kernel evaluates $\mathit{34}.\mathrm{asRat} = (17 + 19) - (2 + 0)$ to `True`.

Trust composition. The kernel-minted output for this claim depends on: (i) the four runtime `Attested_T` payloads (one per tool call) — and *only* those (Theorem 3.1); (ii) the per-source `stats_sum_strict` lift, audited once for this table; (iii) the kernel's native arithmetic check for the `PLift` $= 34$ identity. The Sigma witness "34" surfaces as the certified surface answer; the generated Lean file is the replay artifact (any auditor holding the kernel + lifts + tool adapter can re-typecheck without re-running the LLM). This is the single complex example we use in Sections 3 and 4 to demonstrate that the architecture scales beyond single-shot lookups: composition over multiple lifts preserves the kernel's mint discipline and, by construction, the verified-output guarantee.

P. Full proofs of the safety theorems

This appendix provides the full proofs deferred from Section 3.6. Notation follows the main text: given a VERIFIED run of RUNTIMEVERIFY(s, c) returning the proof object $(\mathit{leanFile}, \mathit{evidence}, \mathit{steps})$, we write *proof* for the term bound by `claim_proof` inside *leanFile* (or `claim_refutation` for refuted claims), $\mathcal{A}(\mathit{leanFile})$ for the set of axioms declared in *leanFile*, and $\mathrm{deps}(\mathit{proof}) \subseteq \mathcal{A}(\mathit{leanFile})$ for the dependency closure of *proof* as reported by `#print axioms claim_proof`.

P.1. Operational whitelist predicate and guarantee boundaries

Whitelist-compliant module. Let *leanFile* be the module text returned by a run, let *proof* be the term bound by `claim_proof` (or `claim_refutation` for refuted claims), and let $\mathrm{deps}(\mathit{proof})$ denote its dependency closure as reported by `#print axioms claim_proof`. We say *leanFile* is *whitelist-compliant for source* s iff every element of $\mathrm{deps}(\mathit{proof})$ lies in one of four buckets: **B1** the fixed EG-VAR prelude (committed at the repository hash in Section G); **B2** the standard Lean 4 core axioms (`Classical.choice`, `Quot.sound`, `propext`); **B3** runtime-emitted observation leaves (`obsN_q,p,att,tag,shape,l1,row,cell,id,max,min`; `csN_q,att`; `tableTool/statsTool`); **B4** per-source lifts whose names match `lift_[0-9a-f]+_.+` drawn from $\Lambda(s)$. Additionally, $\mathrm{deps}(\mathit{proof})$ must contain no `sorry/sorryAx/Lean.Axiom.sorry` and must contain a trust-artifact witness: either `mkVerified` (attestation path), or at least one **B3** observation-leaf together with at least one **B4** per-source lift (direct-evidence lift path, per Section 3.2). The standalone script `verify_axioms.py` evaluates this predicate mechanically on *leanFile*.

Generation-time gate versus replay gate. The kernel call inside `run_pipeline` (`lake env lean`) is the primary safety gate: a claim is emitted as VERIFIED only if the kernel returns a zero exit status on *leanFile*. The replay audit `verify_axioms.py` is a standalone artifact-level check: it accepts *leanFile* as input, invokes the Lean elaborator on its `#print axioms` directive, and enforces the whitelist predicate on the reported dependency closure. Any third party holding *leanFile* and the Lean toolchain can re-run `python -m runtime.verify_axioms` without invoking any LLM role, reproducing the audit decision. The orchestrator (`src/python/runtime/orchestrator.py`) chains the two: a claim is published as VERIFIED only if both calls succeed.

What the theorems do and do not guarantee. Theorems 3.1 and 3.2 guarantee that every VERIFIED output descends from a real tool call and every accepted proof step is type-correct. They do *not* guarantee: (i) that *goalType* is a semantically faithful rendering of the natural-language claim — this is the formalizer's responsibility, evaluated separately by the answer-match axis (Section 5.1(i, iii)); (ii) that the per-source lifts $\Lambda(s)$ correctly interpret the tool's L1 outputs as L2 world facts — a semantically wrong but audited lift can still certify a wrong formalized claim (Section 5.1(ii)). Both are honest trust-artifact boundaries, not architectural oversights: $\Lambda(s)$ is LLM-assisted, curator-reviewed, and frozen per commit (Section 3.6). World-inconsistency containment follows from the `WProp` $\not\to$ `Prop` phase separation in the current Lean encoding; we treat it as implementation detail, not as a main-text theorem (see Section 5.1).

P.2. Lemma chain for the theorems

Lemma P.1 (Checker fidelity)**.** *If* KERNELCHECK*(leanFile) returns* $(\mathrm{True}, \mathit{output})$*, then proof satisfies proof* : *goalType in Lean 4's dependent type theory relative to* $\mathcal{A}(\mathit{leanFile})$.

Proof. The procedure KERNELCHECK(*leanFile*) is implemented at `pipeline.py:4428` by invoking `lake env lean` on *leanFile* and inspecting the elaborator's exit status and output. The Lean 4 kernel is a type-checker for the Calculus of Inductive Constructions: a closed expression t elaborates at type T iff the kernel applies a finite sequence of typing-rule applications that witness $t : T$ starting from the declared axioms and definitions of the environment (de Moura & Ullrich, 2021). The kernel rejects any term that invokes an undeclared identifier, applies a constructor at a non-matching type index, or whose strict positivity or termination constraints are violated. The declared axiom set used during elaboration is exactly the union of (a) axioms from imported modules (the EG-VAR prelude loaded via the `import` preamble at the top of *leanFile*) and (b) axioms introduced by top-level `axiom` declarations in *leanFile* itself; call this union $\mathcal{A}$. If the elaborator reports success, then by the kernel's type-correctness invariant the stored proof term *proof* satisfies *proof* : *goalType* under $\mathcal{A}$. □

Lemma P.2 (Runtime emission discipline)**.** *Every observation-leaf axiom declaration in leanFile — every top-level `axiom` declaration whose name matches one of the **B3** schemas — was emitted by the runtime only after a successful call* EXECUTE(t, q) *whose resulting triple* $(t, q, \mathit{payload})$ *was appended to evidence before the corresponding emission step. The two tool-identifier axioms `tableTool` and `statsTool` are declared once in the module preamble as pure identifiers of the `ToolId` sort and carry no payload content.*

Proof. We prove the claim by exhaustive inspection of the module-emission path in `src/python/runtime/pipeline.py`. The text of *leanFile* is assembled by the function `_build_lean_file` (`pipeline.py`:3174–3488); no other code path emits module text. We enumerate the branches of `_build_lean_file` at which an `axiom` line is appended to the module.

site (file:line)	source	axiom families emitted
3203–3204	preamble	`tableTool`, `statsTool`
3231–3232	common-sense	`csN_q`, `csN_att`
3272–3275	stats (argmax/argmin)	`obsN_row`, `obsN_max`, `obsN_cell`
3318	stats (value-only L1)	`obsN_l1`
3352	stats (topk-by L1)	`obsN_l1`
3387	stats (topk/bottomk)	`obsN_l1`
3399–3404	stats (legacy shape)	`obsN_q,p,att,tag,shape`
3431–3434	table (cell-lookup)	`obsN_row`, `obsN_cell`, `obsN_id`
3442–3447	table (cell-lookup legacy)	`obsN_q,p,att,tag,shape`

Table 9. Axiom-emission sites inside `_build_lean_file`. Each `obsN_*` or `csN_*` branch executes only if the preceding loop variable carries a well-formed payload, which is appended to *evidence* by the runtime tool-dispatch layer before `_build_lean_file` is invoked. The preamble row declares `tableTool`/`statsTool` once per module as pure `ToolId` identifiers.

Each stats and table row lives inside a branch guarded by an evidence entry whose `source` field is `"stats"`, `"table"`, or `"common_sense"`. The tool-dispatch layer and the runtime solver loop in `run_pipeline` append an entry to *evidence* only after a successful EXECUTE(t, q) call returns a well-formed payload; failed or rejected calls return `None` and produce no evidence entry. Therefore, at the point `_build_lean_file` emits any observation-leaf axiom for evidence index i, the corresponding triple $(t, q, \mathit{payload})$ is already present at position i of *evidence*.

A grep over `pipeline.py` for `"axiom obs"`, `"axiom cs"`, `"axiom tableTool"`, and `"axiom statsTool"` recovers exactly the lines listed in Table 9 (together with solver-prompt skeleton lines at `pipeline.py:2943--3080` that build the *prompt string* shown to the LLM solver but do not append to *leanFile*). The prompt-skeleton sites are outside `_build_lean_file` and do not contribute to the module text. □

Lemma P.3 (Reserved names in per-source lifts)**.** *For leanFile to be whitelist-compliant, every top-level declaration in* $\Lambda(s)$ *must have a name matching the regex `lift_[0-9a-f]+_.+`. In particular, per-source lift names must not collide with the identifiers `claim_proof`, `claim_refutation`, `tableTool`, `statsTool`, or any of the schemas `obsN_*`/ `csN_*`.*

Proof. Whitelist compliance places each axiom in $\mathrm{deps}(\mathit{proof})$ into exactly one of **B1**–**B4**. **B1** and **B2** are disjoint from the lift-name regex: the prelude declarations are named after the EG-VAR vocabulary in `src/lean/EGVar/*.lean`, and the three Lean core axioms are fixed names. The runtime-emitted schemas **B3** have forms `obsN_*`, `csN_*`, `tableTool`, `statsTool` that the regex `lift_[0-9a-f]+_.+` does not match. A per-source lift whose name collided with a reserved identifier would either (a) shadow a prelude declaration, causing elaboration failure at `lake env lean` invocation (rejected by Theorem P.1); (b) match a **B3** regex, causing `verify_axioms.py` to classify the axiom in the wrong bucket (a curation-audit failure the regex-conforming prefix rules out); or (c) collide with `claim_proof` / `claim_refutation`, triggering a

declaration-name clash at elaboration time. The enforced `lift_<sourceHash>_<suffix>` prefix mechanically excludes all three. □

Proof of Theorem 3.1. We proceed in three steps.

Step 1: the proof type-checks under the declared axiom set. The verdict VERIFIED is produced at Algorithm 1, line 17, only if KERNELCHECK(*leanFile*) returned $(\mathrm{True}, \textit{output})$ on line 15. By Theorem P.1, this implies $\textit{proof} : \textit{goalType}$ in Lean 4's type theory under $\mathcal{A}(\textit{leanFile})$, and in particular $\mathrm{deps}(\textit{proof}) \subseteq \mathcal{A}(\textit{leanFile})$.

Step 2: the closure is restricted to the whitelist buckets. Whitelist compliance of *leanFile* (the theorem's hypothesis) is the predicate that every element of $\mathrm{deps}(\textit{proof})$ lies in one of the buckets **B1**–**B4**, contains no `sorry`/`sorryAx`/`Lean.Axiom.sorry`, and contains a trust-artifact witness. The implementation at `src/python/runtime/verify_axioms.py` parses the `#print axioms` output and enforces each clause directly; Theorem P.3 makes the bucketing unambiguous.

Step 3: observation-leaf axioms in the closure are evidence-backed. Partition $\mathrm{deps}(\textit{proof})$ into four subsets $D_{\mathbf{B1}}, D_{\mathbf{B2}}, D_{\mathbf{B3}}, D_{\mathbf{B4}}$ by bucket. Step 2 covers the closure.

For every $\omega \in D_{\mathbf{B3}}$: the axiom ω was declared in *leanFile* by one of the emission sites tabulated in Table 9. By Theorem P.2, the emission was preceded in time by the runtime appending a corresponding triple to *evidence* (for `obsN_*`/`csN_*` axioms); the two `ToolId` axioms `tableTool` and `statsTool` are explicitly admitted without an evidence entry because they carry no payload. Hence every observation-leaf axiom in $D_{\mathbf{B3}} \cap \mathrm{deps}(\textit{proof})$ is backed by an entry of *evidence*.

For $D_{\mathbf{B1}}$, each axiom is declared in one of the fixed EG-VAR prelude files (`EGVar/Basic.lean`, `EGVar/TableL1.lean`, `EGVar/WorldOntology.lean`, `EGVar/Tactics.lean`), committed at the repository hash reported in Section G.

For $D_{\mathbf{B2}}$, each axiom is one of `Classical.choice`, `Quot.sound`, or `propext` from Lean 4's core library.

For $D_{\mathbf{B4}}$, each axiom has a name matching `lift_[0-9a-f]+_.+` with hex prefix equal to the MD5 of s; the whitelist predicate enforces this regex, and the full set of such axioms is committed to `data/tier1_table_formalizations.json` at the reported commit hash.

Combining steps 1–3, any $\omega \in \mathrm{deps}(\textit{proof})$ is either (a) a prelude axiom from $D_{\mathbf{B1}}$, (b) a standard Lean core axiom from $D_{\mathbf{B2}}$, (c) a runtime-emitted observation-leaf backed by a prior tool execution in *evidence*, or (d) an audited per-source lift from $\Lambda(s)$. □

Proof of Theorem 3.2. An "accepted inference step" is, by the definition of the runtime's VERIFIED verdict (Algorithm 1, line 17), one that appears in the term *proof* returned by KERNELCHECK(*leanFile*) with $\mathrm{accepted} = \mathrm{True}$. By Theorem P.1, *proof* type-checks under the axiom set declared in *leanFile*. The kernel's type-correctness invariant requires that *proof* is constructed by a finite, typing-rule-compliant sequence of inference steps leading to $\textit{proof} : \textit{goalType}$; consequently every such inference step is valid in Lean 4's dependent type theory relative to the declared axioms. □