CausalForge: A Formally Grounded, Self-Improving Agentic Framework for Automated Research in Causal Inference

Jiyuan Tan Vasilis Syrgkanis

Abstract

Automating theoretical research is constrained not only by the generation of candidate results, but also by their reliable evaluation. A common approach is to close the research loop with a large language model (LLM) reviewer. However, such reviewers remain empirically unreliable: they may accept fabricated papers and detect them at rates close to chance [19]. We present CausalForge, a framework for automated theoretical research in causal inference grounded in the Lean proof assistant. CausalForge combines Causalean, a foundational Lean library for causal inference containing 7,035 machine-checked declarations developed with language-model assistance under human design and review, with CausalSmith, a self-improving agentic pipeline that selects research topics, proposes results, formalizes statements, constructs proofs, and presents the resulting artifacts for human inspection. Because a machine-checked proof establishes only that a formal statement follows from its assumptions—not that the statement faithfully captures the intended scientific claim—the pipeline augments kernel verification with a statement audit that compares each formal theorem against the informal claim it is intended to express. We evaluate the system using artifacts produced by completed autonomous research runs. The source code, formal library, and run records are available at `https://github.com/Jiyuan-Tan/CausalForge`.

1 Introduction

Language models can now generate research artifacts—conjectures, proofs, experiments, and complete papers—more rapidly than they can be evaluated. In empirical fields, this primarily increases the burden of review. In theoretical work, it presents a more direct risk: an incorrect theorem may be indistinguishable from a correct one until it is verified by an expert, but expert verification is slow and costly. Many systems for automated mathematical and theoretical research address this bottleneck by delegating evaluation to another language model: one model proposes and another reviews, with the review model supplying the main evaluation signal.

Empirical evaluations already show the defects of LLM reviewers. Independent evaluations of the AI-Scientist systems [49] report hallucinated numbers, frequent coding failures, and little genuine novelty [5]. More concerningly, Jiang et al. [19] show that a calibrated board of LLM reviewers accepts deliberately fabricated papers up to 82% of the time while detecting the fabrication at near-chance rates. These findings make LLM review an unreliable evaluator of correctness.

Formal verification provides a different basis for evaluation. When a theorem is formalized in Lean 4 and its proof is accepted by the kernel, the proof is guaranteed to be correct relative to Lean 4's trusted core. This guarantee is unavailable to an LLM reviewer and removes proof soundness from the model's judgment. However, simply asking an LLM to verify its claims in Lean 4 runs into two obstacles: cost and faithfulness.

The first obstacle is cost. Formalizing a result from first principles is slow and labor-intensive. Causal inference needs a reusable library of identification theorems, estimators, and asymptotic

results. Such a library lets each proof begin from verified causal infrastructure, which spares every run the work of rebuilding standard theory from scratch. The second obstacle is the match between the formal theorem and the intended claim. Passing the type checker establishes proof validity while leaving this match to be checked. A statement can be free of `sorry` and still be mathematically uninformative: a definition may reduce to `True`; or an unproven step may be introduced as an `axiom` that the kernel accepts. Recent work documents this gap from several angles [9, 17, 30], and we observed it in our own runs as well. In one, an agent converted difficult lemmas into `axioms`, yielding a type-checking shell with axioms substituting for proof. In another, the model altered a hypothesis, so the Lean 4 code proved a statement different from the one the paper claimed. Both cases are kernel-acceptable; detecting these failures requires assessing the statement as well as the proof. The central question is therefore whether the formal statement expresses the intended claim. Our pipeline answers it with a fine-grained audit step (Section 5.3) that compares each formal statement with the claim it is meant to express, so that the Lean 4 development faithfully represents the natural-language result. Together with the kernel, this audit supports a two-part guarantee: the proofs the pipeline produces are machine-checked, and the statements they establish are the ones the paper reports.

We instantiate this approach in causal inference through CausalForge, which has two components. Causalean is a Lean 4 library covering the causal-inference toolkit, giving agents verified primitives to compose. It was itself built with language-model assistance: humans chose what to formalize and how to state it, agents drafted the definitions, proofs, and docstrings under those decisions, and a human then reviewed the resulting Lean 4 statements. CausalSmith is a self-improving agentic pipeline that selects or accepts a research topic, proposes a result, formalizes it, proves it, and presents it. The pipeline represents each result as a logic graph whose nodes are statements, checks each node against the intended claim, and promotes missing lemmas into Causalean once they have been proved and reviewed. The system distinguishes two layers of trust: the kernel establishes proof soundness; the statement audit checks whether the formal statement matches the intended claim.

Contributions.

- Causalean, a broad Lean 4 formalization of causal inference spanning graphical and structural causal models, potential outcomes and identification, panel methods, experimentation, estimation, and statistical theory. It comprises 7,035 machine-checked declarations, was written with language-model assistance under human design, and provides a retrieval interface for agents (Section 4).
- CausalSmith, an end-to-end pipeline whose Discovery stage can select its own research topic and propose a causal-inference result, which the pipeline then formalizes, proves, and presents. It also contains a library feedback loop that grows Causalean with the reusable lemmas and theorems its runs demand (Sections 5 and 5.3).
- An evaluation drawn from 123 recorded runs: a catalogue of machine-checked results and a headline result found by the system, which closes a gap in Zeng et al. [54] (Section 6). The catalogue also exposes an asymmetry in what the system proposes well: when it chooses its own question, its accepted results are concentrated in questions where the literature has already located the gap and the missing work is technical, while questions whose contribution would have to be a new idea are downgraded for reducing to known constructions (Section 6.2).

Availability. The Causalean library, the CausalSmith pipeline, and the run record backing Section 6 are available at `https://github.com/Jiyuan-Tan/CausalForge`. A companion site at

Table 1: Where CausalForge sits among neighboring systems. "Statement audit?" asks whether a system checks that the formal statement means the intended claim, beyond type-checking.

System	Proposes theorem?	Machine-checked proof?	Statement audit?	Evaluator soundness	Self-improving?
AI Scientist [49]	✓	—	—	LLM judge	—
FunSearch / AlphaEvolve [32, 36]	partial	✓	—	task metric	✓
AlphaProof [16]	—	✓	n/a	sound	partial
Causal benchmarks [20, 21]	—	—	—	fixed labels	—
Causal agents [46]	—	—	—	none	—
CausalForge (ours)	✓	✓	✓	sound + audit	✓

`https://jiyuan-tan.github.io/CausalForge/` provides a browsable view of the library, including the natural-language statement of each declaration and the generated write-up of each accepted result.

Table 1 places CausalForge against the systems it invites comparison with. Automated theorem provers prove statements they are handed; automated-research agents propose statements and leave theorem verification outside their core workflow; benchmarks test causal competence against fixed labeled answers. CausalForge contributes the combination of proposing a causal theorem and auditing that its formal statement means what was claimed.

2 Related Work

Autoformalization, theorem proving, and statement matching. A large body of work uses language models with proof assistants, from neural proof search [24, 34] and early autoformalization [18, 48] to whole-proof generation and repair [11], retrieval-augmented provers [35, 50], and competition-level systems [16, 43]. These systems are usually measured on benchmarks of known problems, including MiniF2F, ProofNet, and PutnamBench [4, 44, 55]. Our setting lacks the fixed reference on which these methods rely: because the theorems are new, there is no gold statement or proof against which to evaluate them. This precludes reference-equivalence scoring and requires an explicit assessment of whether the formal statement matches the intended claim. Recent studies demonstrate that passing the kernel is not equivalent to stating the intended theorem—`sorry`-free formalizations still fail expert review [17], type-correct statements can be semantically wrong [9, 30], and vacuity and reward hacking are common enough to benchmark [45]. Proposed remedies include roundtrip and back-translation equivalence checks [2, 26, 31], learned alignment scorers between a statement and its formalization [28], and a rejection of surface-overlap metrics as evidence of a correct statement match [33]. We operationalize these findings as an audit within a discovery loop, localized over a proof-dependency graph in the style of Lean blueprints and proof-flow tools [6, 56]. The broader shift toward proof-assistant-grounded reasoning [51] provides the context in which this system operates for causal inference.

Automated research agents and verified discovery. Agentic systems that carry out end-to-end research [49] are vulnerable when an LLM judges their output, whether by independent re-evaluation [5] or by adversarial construction [19]. A complementary tradition grounds discovery in a hard verifier—program search against a fitness function [32, 36] or a proof against a kernel. Between these poles sit hypothesis-generating "co-scientist" agents that propose and refine scientific claims but validate them empirically rather than formally [14]. We use a hard verifier for proof

soundness and add an audit of whether the specification is the intended one, a task that their fixed specifications do not require.

LLMs for causal inference. A parallel literature examines whether language models can reason causally. It divides into three strands, each treating causal reasoning as something to test or deploy rather than something to prove; recent surveys organize this work by causal task and intervention level [29]. Benchmarks measure whether models answer causal queries or choose valid research designs—CLadder and Corr2Cause on graphical and correlational reasoning [20, 21], CaLM and QRData on broad and data-grounded causal question answering [7, 27], and, closest to our own domain, benchmarks that target end-to-end causal inference on real scientific studies and disentangle identification from estimation [1, 37]. Across these benchmarks, surface accuracy can conceal shallow, retrieval-driven reasoning. A critical strand finds that models can explain causal language while hallucinating the reasoning [12], recite memorized causal facts rather than reason interventionally [53], commit elementary causal fallacies [22], and are flattered by benchmarks solvable through knowledge lookup [52]. A third strand builds agents that do causal analysis, automating discovery, data preparation, and estimator selection but offering no correctness guarantee [8, 23, 38, 46]. Closest to us in form are agentic systems for causal inference itself: end-to-end pipelines that map a dataset and question to a chosen estimator and result [3], LLM-assisted search for instrumental variables [15], and the multi-agent IV Co-Scientist [39], which proposes, critiques, and refines candidate instruments. These share our propose-and-critique structure but validate their output statistically or empirically; none produces a machine-checked causal theorem. Work pairing language models with formally verified causal inference appears to be limited to theorem-proving systems that verify general mathematics rather than causal inference [40] and to the causal-LLM line above, which does not provide formal verification. CausalForge addresses this intersection.

The closest related systems. Two efforts are particularly closely related. A Lean library for economics [13] formalizes established economic theory, and a multi-agent system formalizes asymptotic statistical theory with auditor and reviewer agents that already guard against vacuity [47]. CausalForge differs in three respects: it discovers novel results rather than formalizing known ones; it audits statement–claim matches node by node over an explicit logic graph, so a change re-opens only the affected nodes; and it grows its reusable library by promoting proved lemmas for later runs. Although anti-vacuity auditing is established prior art, graph-localized, incremental auditing within a self-improving discovery loop is the contribution advanced here.

3 Background

Causal inference. Causal inference concerns the consequences of intervention rather than observation alone. Two frameworks are widely used in the field: structural causal models (SCMs) and potential outcomes (PO). An SCM models causal relationships with a directed acyclic graph and represents interventions with the do-operator; its classical results include the graphical identification criteria delivered by do-calculus and the ID algorithm. Potential outcomes instead attach to each unit a family of counterfactual responses, one per treatment level, and write causal estimands—the average treatment effect, the effect on the treated, difference-in-differences, the local average treatment effect—as functionals of their joint distribution. An estimand is identified when it is a function of the observational distribution alone and only partially identified when the data pin it to a set rather than a point, as with Manski or Balke–Pearl bounds. Causalean (Section 4) formalizes most classical results under these frameworks.

Lean 4 as a proof checker. Lean 4 is an interactive theorem prover and programming language in which mathematics is written in a fully formal language [10]. A definition is a term, a proposition is a type, and a proof of that proposition is a term of that type, so checking a proof reduces to type-checking a term. A user does not write such terms directly: they write *tactics*—commands such as `intro`, `simp`, or `induction` that manipulate an explicit goal state—and the system elaborates them into a proof term. Mathlib, Lean 4's community mathematics library, supplies the analysis, probability, and measure theory that a causal-inference development builds on [42].

What makes this useful as an evaluator is where the trust sits. The elaborator, the tactic language, and any model that wrote the tactics are all untrusted; only a small kernel decides whether the final term has the claimed type. When the kernel accepts a term of type τ, τ is provable in Lean 4's underlying type theory, and the kernel also reports which axioms the proof depends on. We call this property proof soundness. It is a guarantee about the proof and not about the statement: whether τ is the proposition the researcher intended to prove is a separate question, which Section 5.3 takes up.

4 The Causalean Library

The cost of formalizing results from first principles limits automated discovery, and Causalean supplies the reusable foundation that discovery needs. It is a foundational Lean 4 library of causal inference that gives agents verified results to compose across runs. For example, a proof that requires the backdoor adjustment formula or the asymptotic normality of a debiased estimator can reuse a result that has already been stated and proved. This section describes the library's contents and the design properties—breadth, stability, and searchability—required by a research pipeline.

4.1 Design and scope

Causalean holds paper-agnostic mathematics: the definitions and theorems any causal result might reuse. One-off lemmas of a particular paper live instead in the pipeline package, and continuous integration enforces the separation: runs read from the core freely but write back to it only through the explicit human promotion step in Section 5.4.

The library was built with LLM assistance rather than written by hand, and the division of labor follows the trust model of Section 1. Humans set the scope, choose which results to formalize, and fix the definitions that later theorems are stated against; agents then draft the statements, proofs, and docstrings, iterating against the compiler; and a human reads the resulting Lean 4 statements before they are kept. Nothing enters the library until the kernel accepts it and it survives the same unproved-shortcut screen the pipeline applies to its own output (Section 4.5). The human then checks that the formal statement matches the intended claim and that it is stated in a standard form.

Two smaller conventions keep the library usable by an agent: each declaration's prose lives once in its Lean 4 docstring, from which the human-readable API is regenerated, so documentation stays aligned with code; and files stay small and hold a single topic, so a retrieved declaration arrives with enough surrounding context to apply it.

The compiled environment index records 7,035 declarations: 4,616 theorems, 2,015 definitions, and 404 structures, instances, and inductive types, across 973 files and roughly 262,000 lines (Table 2).[1] Ten clusters cover the field from graphical foundations to asymptotic statistics.

[1]These repository statistics are a snapshot. The totals may increase because the CausalSmith pipeline runs continuously and can promote newly proved, reusable declarations to Causalean.

Table 2: CAUSALEAN coverage by cluster. All figures are read from the compiled environment index (`lake exe library_index`); the per-cluster kind columns sum to the grand total. "s/c/i" abbreviates the remaining declaration kinds—structures and classes, inductives, and instances.

Cluster	Files	Lines	Defs	Thms/Lem.	s/c/i
Estimation	168	54,246	298	621	100
Stat	202	47,165	213	894	38
PO	138	41,415	592	947	80
SCM	88	36,342	208	523	67
Mathlib (local)	113	24,778	104	500	11
Panel	68	18,382	258	389	44
Experimentation	101	16,095	172	370	12
Graph	21	10,866	70	160	28
ML	50	6,330	56	100	13
Discovery	24	6,321	44	112	11
Total	**973**	**261,940**	7,035 **declarations**		

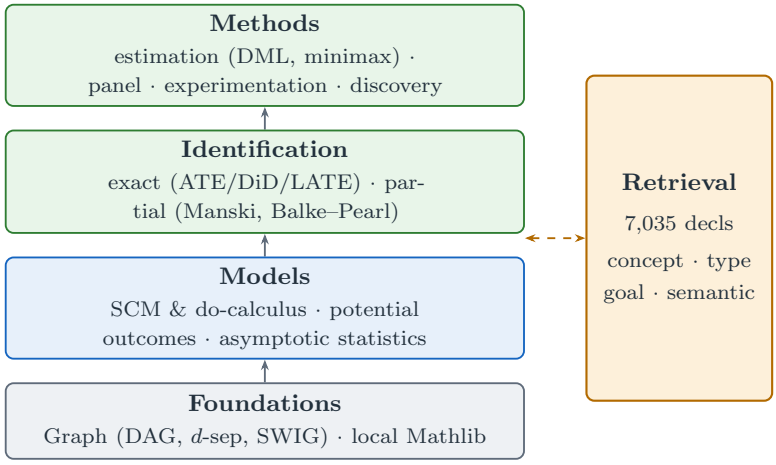

Figure 1: CAUSALEAN in layers: graphical and measure-theoretic foundations support the model languages, which support identification, which supports the estimation and design methods on top. A retrieval index over all 7,035 declarations spans the stack and is the interface the pipeline queries.

4.2 A tour of the clusters

The library layers from graphical and measure-theoretic foundations up to the estimation and design methods that depend on them (Figure 1).

Graphs and structural models. The Graph cluster builds directed acyclic graphs with a decidable edge relation and a stored topological order, on top of which sit the parent, child, and ancestor operations, *d*-separation implemented as Bayes-Ball reachability, single-world intervention graphs, and the *c*-component decomposition that the identification algorithm needs. The SCM cluster turns these graphs into structural causal models. It defines interventions and the do-operator, proves the semi-graphoid axioms that justify do-calculus, represents factored kernels, and develops graphical identification proper: backdoor and frontdoor adjustment, general adjustment criteria, and a soundness proof for the ID algorithm on the discrete positive class. This is the part of the library where a causal query becomes a formula in the observational distribution, or receives an

explicit nonidentification certificate.

Potential outcomes and identification. The PO cluster is the largest by definitional surface, because it carries the two identification branches the pipeline uses most. Exact identification covers the standard estimands—ATE, ATT, difference-in-differences and its Callaway–Sant'Anna group-time refinement, LATE, regression discontinuity, proximal and dynamic-treatment designs—each stated as an equality between a counterfactual contrast and an estimable functional. Partial identification covers the bounds one falls back on when point identification is impossible: Manski's worst-case bounds with their monotone-treatment-response and instrument refinements, the sharp Balke–Pearl bounds for a binary instrument, Lee's trimming bounds under selection, marginal-sensitivity models, and Imbens–Manski inference for the resulting sets.

Estimation and asymptotic statistics. Estimation, the largest cluster by volume, is the semiparametric machinery: double/debiased machine learning for the ATE, ATT, and CATE; efficient influence functions obtained by projection onto a tangent space; structure-agnostic minimax lower bounds with matching estimators; and convergence rates for nonparametric instrumental variables. These results draw on the Stat cluster, which formalizes the probability and empirical-process theory underneath: central limit theorems, U-statistics with their Hájek projections, Glivenko–Cantelli and bracketing-entropy tools, M- and Z-estimation, concentration inequalities, and the bootstrap. Because these results live in the library, an estimation proof can cite a limit theorem directly and focus on the causal argument.

Panel, experiments, and discovery. The remaining clusters round out the toolkit. Panel characterizes the estimands that two-way fixed-effects and difference-in-differences regressions actually recover, including the negative-weights decomposition and event-study contamination that motivate modern DiD estimators. Experimentation develops experimentation theory, including randomization inference, Horvitz–Thompson estimation, Neyman allocation, and central limit theorems under network interference. Discovery formalizes identifiability results for causal structure learning, such as LiNGAM under non-Gaussianity and invariant prediction, while ML and the local Mathlib supplements provide the learning-theory and measure-theoretic lemmas the rest of the library draws on.

4.3 Flagship results

The library also provides a collection of classical flagship results from the literature. Table 3 catalogues the principal substantive result families across the library's major research areas. Each entry is a named declaration with a machine-checked proof; Section A gives its source location. Curators have also marked 1,597 library theorems as headline results and recorded 3,677 statement-level review stamps, each a human assessment of whether the formal statement matches the intended claim.

Table 3: Flagship theorems across CAUSALEAN's major research areas. Each entry is a named, machine-checked declaration; long identifiers may wrap.

Result	Declaration
Graphical and structural causal models	

Table 3 continued

Result	Declaration
Markov equivalence iff two DAGs have the same skeleton and immoralities	`markovEquiv_iff_sameSkeleton_sameImmoralities`
Rule 2 of do-calculus under its graphical condition	`do_rule2_kernel`
Backdoor adjustment identifies an interventional distribution almost everywhere	`backdoor_identifiable_ae`
Frontdoor adjustment identifies an interventional distribution almost everywhere	`frontdoor_identifiable_ae`
Soundness of the graphical ID algorithm on the discrete positive class	`id_sound_discrete`
Causal discovery	
LiNGAM identifiability from nonzero source kurtosis	`lingam_identifiability_kurtosis`
Soundness of invariant causal prediction	`icp_sound`
Completeness of invariant prediction for linear-Gaussian models	`icp_complete_linearGaussian`
Identifiability in linear causal disentanglement	`disentanglement_identifiability`
Potential outcomes, exact identification, and partial identification	
Wald-ratio identification of the local average treatment effect	`late_wald`
Difference-in-differences identification of the ATT	`att_did`
Callaway–Sant'Anna group-time ATT identification	`att_csdid`
Sharp regression-discontinuity identification	`rdd_identification`
Fuzzy regression-discontinuity identification	`frd_identification`
Wald identification for dynamic treatment timing	`whenToTreat_wald`
Manski worst-case ATE bounds	`manski_bounds_ATE`
Sharpness of the Balke–Pearl bounds for a binary instrument	`balkePearl_sharp`
Lee bounds for the ATT under selection	`lee_bounds_ATT_AS`
Pointwise Imbens–Manski coverage for partially identified parameters	`imbensManski_pointwise_coverage`
Estimation and statistical theory	
Asymptotic normality of the debiased-ML ATE estimator	`dml_ATE_tendstoNormal`
Attainment of the Hahn efficiency bound by debiased-ML ATE	`dml_ATE_attains_hahn_bound`
Asymptotic linearity of debiased-ML ATT estimation	`dml_ATT_isAsymLinear`
Asymptotic normality of partially linear DML	`plr_dml_tendstoNormal`
Asymptotic linearity of sequential doubly robust DTR estimation	`seqDR_dml_isAsymLinear`
Structure-agnostic minimax lower bound for ATE estimation	`minimax_lower_bound_var_causal`
Optimal weighting for generalized method of moments	`gmm_efficiency`

Table 3 continued

Result	Declaration
Central limit theorem for regular order-m U-statistics	`uStatisticOrder_clt`
Panel and event-study methods	
Causal decomposition of staggered-adoption TWFE into weighted contrasts	`twfe_po_decomposition`
Characterization of linear-unbiased BJS imputation estimators	`bjs_linear_unbiased_iff_imputation_form`
Event-study pretrends induced by post-treatment effects	`apparent_pretrends_from_post_treatment_of_cellGrid`
Experimentation under interference	
Consistency of Horvitz–Thompson estimation under unknown interference	`htEst_consistent_eate`
Central limit theorem for a two-stage interference direct effect	`directEffect_clt`
Stein-method CLT for estimators under network interference	`localDependenceCLT_of_stein`
Wald coverage under network interference	`wald_coverage_of_stein`

4.4 Retrieval

A library of this size requires effective declaration retrieval, so retrieval is a primary component of Causalean. During the build, every declaration is elaborated into an index that records its name, kind, module, source text, docstring, cross-references, axiom dependencies, and whether its proof uses `sorry`. A companion tier embeds all 7,035 declarations with a 1,024-dimensional sentence encoder. The search engine provides three modes over this index: a concept mode that expands a natural-language query with causal-inference synonyms, a type-pattern mode for structural queries, and a goal-directed mode that ranks candidates against an open proof goal. Each mode can fuse its lexical ranking with the embedding tier and supports cluster and module filters. This interface lets the pipeline reuse verified results as composable proof ingredients.

4.5 Axiom checks and positioning

The library passes the axiom checks required for its use as a verified foundation. All 7,035 declarations are `sorry`-free, and the corpus is free of hand-written `axiom`; the only non-standard axioms are compiler artifacts introduced by `native_decide`, which occur in finite-graph decidability arguments and several minimax calculations and are reported explicitly. Two results are not original to Causalean but adapted, re-licensed copies of external Lean 4 developments, bumped to Causalean's Lean 4/Mathlib pin: the Karush–Kuhn–Tucker first-order necessary conditions under LICQ and affine constraint qualifications, from OptSuite's `optlib` [25] and consumed by the Estimation cluster's minimax lower bounds; and Rademacher complexity, McDiarmid's inequality, symmetrization, and the Dudley entropy integral, from `lean-rademacher` [41] and consumed across Stat, Estimation, and ML. Two recent libraries provide relevant points of comparison—a Lean formalization of economics and a multi-agent formalization of asymptotic statistics (Section 2). Causalean differs by covering the causal-inference toolkit and by serving as a library that the

research pipeline can compose and extend, as described in the next section.

5 The CausalSmith Pipeline

CausalSmith operationalizes the library within a research pipeline. There are four stages in our pipeline: Discovery, Formalization, Proof Construction, and Presentation. The pipeline either accepts a researcher-supplied topic or selects one before proposing a causal-inference result in natural language. The pipeline then formalizes the result, proves it in Lean 4, and produces a paper linked to the formal development. Once the Discovery stage fixes the intended claim, the central object throughout the pipeline is a logic graph. Discovery creates this graph; formalization maps its nodes to planned Lean 4 declarations; proof construction fills those declarations and updates their dependencies; statement matching checks each formal node against the intended claim; and presentation reads the reviewed graph when it writes the paper.

Essentially, CausalSmith is a deterministic state machine that executes each stage and retries failures within fixed limits. A lightweight orchestrator launches the state machine, records its verdicts, and requests human input only at a few fixed decision points. In the automatic mode used for most runs, the orchestrator advances through every stage on its own and pauses for a person only at the final acceptance decision, so a typical run completes without a human in the loop. Figure 2 summarizes the workflow; Sections C to F give an operational specification of the stages, artifacts, recovery rules, and presentation workflow.

5.1 The Logic Graph as a Persistent Run Record

Dependency graphs for proofs are established practice; Lean blueprints and proof-flow tools represent developments as DAGs of statements [6, 56]. CausalSmith uses the graph as both a plan and a review record. It turns the global question "is this result proved, and does the formal theorem match the intended claim?" into node-specific questions that can be revisited when the corresponding statements change.

Each result is stored as a graph whose nodes are statements—a setup, a definition, an assumption, a lemma, or the headline theorem—and whose edges record dependencies. A node carries its natural-language statement and its Lean 4 declaration, a review status in {`unreviewed`, `matched`, `derived`, `drift`} recording the current statement match, and a class: a `gated` node represents substrate missing from Causalean and must be discharged by a proof before the result can be recorded as complete, while a `cited` node is a borrowed result with source evidence that sits off the critical path. Assumptions carry a finer label still—claim refinement, regularity bookkeeping, or library gap—distinguishing a hypothesis that genuinely narrows the claim from one that is only technical scaffolding. Edges come in two kinds, `statement-uses` and `proof-uses`, which separate what a claim means from what its proof consumes; a validator rejects duplicate nodes and malformed dependencies. A typical completed result has on the order of a hundred nodes and a few hundred edges (Figure 3).

Within Discovery, the selector supplies the topic anchor but does not yet create the logic graph. The proposal and solver then write the natural-language nodes and their dependency edges from that anchor. Formalization attaches the intended Lean 4 declarations. Proof construction adds the proof dependencies. Statement matching changes review statuses such as `matched` or `drift`. Presentation then uses the reviewed graph as the source of truth for the paper-to-code crosswalk. The critical path consists of `gated` nodes: a result is complete when every gated node has a proof and is marked `matched`. When a statement changes, the pipeline returns that node to `unreviewed` and rechecks the affected frontier and its dependents.

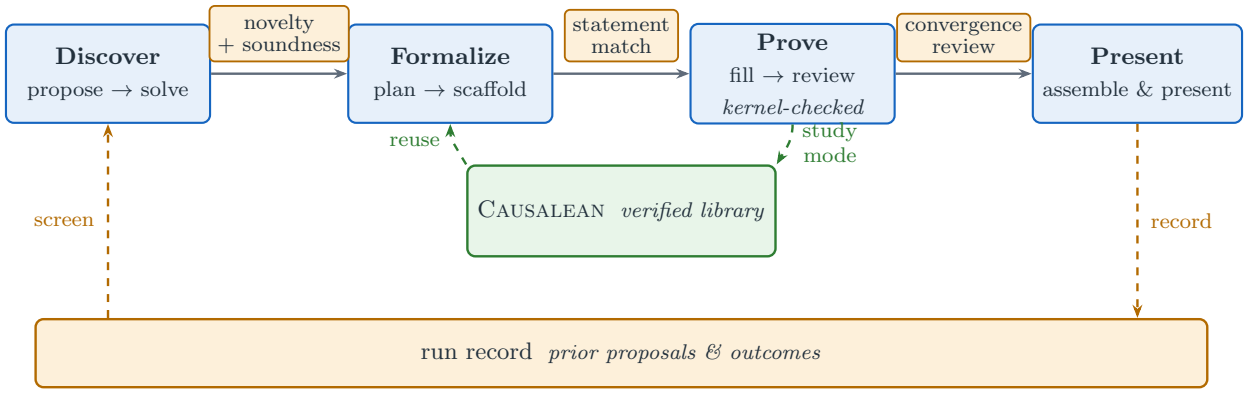

Figure 2: The CAUSALSMITH pipeline. Discovery includes topic selection: when a researcher does not supply a topic, a dedicated selector searches and ranks candidate directions, screens them against the run record, and admits a topic only after an adversarial quality gate. The resulting claim moves left to right through discovery, formalization, proof, and presentation, clearing a gate between stages (novelty and mathematical soundness; statement matching; and a final convergence review). The kernel establishes proof soundness during the proof stage. The convergence review is an independent dual-model review of the full frozen graph, including nodes previously marked as matched; the presentation stage then assembles the reviewed result into a paper. Two feedback channels grow the system (Section 5.4). Stages reuse CAUSALEAN through retrieval, and when a run needs a new load-bearing lemma, study mode proves it and promotes it back into CAUSALEAN. In parallel, every run—accepted, downgraded, or failed—is recorded in the run record, whose entries are consulted during screening before the next proposal is drafted. The logic graph is the object that moves through the diagram: Discovery creates it from the selected or supplied topic, formalization maps it to Lean 4, proof construction updates it, the statement-match and convergence gates review it, and presentation writes from it. See Sections C to F for the operational stage flow (Figure 4), run record and recovery, graph-controlled proof loop, and presentation pipeline.

5.2 Discovery, formalization, and proof

The pipeline supports two entry modes. A researcher may supply a topic anchor, or the main orchestrator may invoke the `causalsmith-topics` module as the first part of Discovery. The selector searches recent work, reads theorem-bearing papers and their citing or follow-up literature in full, and drafts a small slate of directions the literature leaves open. It then checks candidates against active and recorded runs, ranks them by mathematical promise and by whether the result would have a concrete downstream use, and submits the leading candidate to an independent adversarial topic gate. The gate requires a precise, non-vacuous research object, a defensible novelty tier (the pipeline's ordered grade of a result's novelty), and a genuine downstream use before the selector emits the topic anchor, question identifier, and specialization used to continue Discovery. If bounded re-selection cannot produce an accepted candidate, the pipeline stops for operator direction instead of spending a theorem run on an ungrounded topic.

Once an anchor is fixed, the remaining discovery stages derive the result in natural language and build the graph. A proposer drafts a question and an informal solution, a novelty-and-duplication gate compares it with the literature and judges whether it is novel, and a solver derives a formalizable mathematical core in natural language. These outputs become the initial graph nodes and edges. When a proposed claim is too strong, the solver may narrow it to a valid result, but it may not weaken a statement merely to complete a proof or strengthen a hypothesis merely to obtain a result.

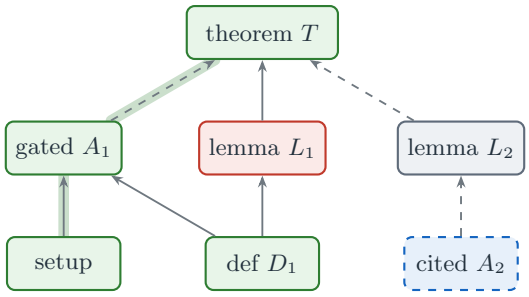

Figure 3: A logic graph (schematic). Nodes are statements; fill encodes review status (**matched**, **drift**, **unreviewed**); a dashed blue border marks a cited node, borrowed and off the critical path. Solid edges are `statement-uses`, dashed edges `proof-uses`. The green underlay traces the critical path of gated nodes the review must clear before final recording.

A sound but insufficiently novel proposal is downgraded, whereas an incorrect or trivial proposal is rejected.

Formalization turns this graph into a proof plan. Each node receives an intended Lean 4 declaration, a module location, and either a proof obligation or a reuse target from CAUSALEAN. The plan is translated into a scaffold with proof obligations. A review-and-fill loop then resolves these obligations. On each iteration, the reviewer first checks that every declaration remains aligned with the graph and repairs only declarations whose statements have changed; it then fills the remaining obligations, using the retrieval interface in Section 4.4 to reuse existing results. A deterministic lint flags hypotheses unused by the proof, which may indicate a vacuous or mis-stated theorem.

5.3 Statement matching

The statement-match gate in Figure 2 is the statement faithfulness audit. The Lean 4 kernel establishes proof soundness, but, as Section 3 notes, it does not establish that the formal statement is the one the researcher meant to prove: the kernel accepts a theorem whose hypotheses are contradictory, a definition that unfolds to `True`, or a lemma asserted as an `axiom` in place of a proof. This gate checks whether each formal statement still matches the intended claim stored in its graph node. The target is logical equivalence: the intended claim and the Lean 4 declaration must have equivalent assumptions and conclusions, with neither statement stronger nor weaker than the other. The two do diverge in practice: a development can be free of `sorry` and still fail expert review because a definition is too narrow, a hypothesis is vacuous, or an unproven step has been promoted to an axiom.

Table 4 lists the failure modes the statement review targets, grouped into four families. Wrong-statement errors compile but state the wrong proposition. Vacuity covers a concept defined as `True`, an unsatisfiable hypothesis, and a witness that carries no real obstruction. Unproved shortcuts cover an unproven step introduced as an `axiom` or through `sorry`, `admit`, or `native_decide`, and a load-bearing node mis-tagged cited. Over-narrow statements cover dropped hypotheses and hardcoded constants that leave the theorem narrower than claimed. Every entry type-checks; the kernel accepts all of them.

Two layers perform the checking. A mechanical scan rejects any completed artifact—and the closure of library modules it touches—that contains `axiom`, `sorry`, `admit`, `native_decide`, `opaque`, or `unsafe`, and a build gate rejects any artifact with an open proof state. By focused review, several agents read the Lean 4 declaration text extracted from the compiled file, compare it against the intended statement, and mark `drift` on any divergence; the review requires every claimed

Table 4: Failures that can pass the kernel while changing the intended claim.

Family	Failure mode	How the review catches it
Wrong statement	Type-checks but states the wrong proposition.	Per-node comparison with the intended claim.
Vacuity	Concept defined as `True`; unsatisfiable hypothesis; witness with no concrete obstruction; a trivially-true side collapsing a biconditional.	Witness normalization drops a witness with no obstruction; hypothesis-satisfiability check.
Unproved shortcut	Unproven step as `axiom`; `sorry`/`admit`/`native_decide`; load-bearing node mis-tagged `cited`.	Mechanical scan for forbidden proof shortcuts; source check for borrowed nodes.
Over-narrow statement	Dropped hypothesis; hardcoded constant where general was intended; over-narrow model class.	Comparison with the intended claim; unused-hypothesis lint.

witness to carry a concrete obstruction and every `cited` node to have a real source match before final recording. A reviewer does not compare the two surface forms directly: it first back-translates the intended claim into a Lean-shaped hypothesis list and conclusion and then compares that against the declaration, following the roundtrip and back-translation checks of the autoformalization literature [2, 26]. No surface-overlap score is used as evidence of a match, a use that literature rejects [33].

After the proof obligations are closed, a dual-model convergence reviewer rechecks the full frozen graph, including nodes outside the current change frontier. This review independently rechecks every statement against the intended claim before the result enters the presentation stage. It is therefore the final gate in Figure 2.

5.4 Library feedback

The graph also controls what the pipeline carries forward after a run. CausalSmith grows through two feedback channels: a run record that preserves what has been attempted, and a verified library that grows with what has been proved.

The run record preserves every run at every disposition, from accepted to rejected, together with its proposal, typed core, and review verdicts (Table 5 and section D). Before a new proposal is drafted, the reconnaissance stage searches these records for open gaps and near-duplicates, and the novelty-and-duplication gate compares a fresh proposal against prior entries. Failed and downgraded runs contribute alongside accepted ones: a documented rejection prevents a later run from repeating the same dead end, and a downgraded result sharpens the novelty target for a related question. Because every run contributes evidence, subsequent discovery starts from an accumulating record of successful and unsuccessful attempts.

The library channel closes the loop for load-bearing lemmas. When the graph identifies a missing lemma—a `gated` node that remains unproved and unsupported by the current library—the pipeline can prove and promote the lemma to Causalean. A library builder executes a bounded `study` workflow: a scaffolder creates a plan and a file with proof obligations; parallel proof-filling agents resolve the obligations against the live compiler; and a reviewer assesses whether the resulting lemma is generic, reusable, non-vacuous, and `sorry`-free. A build gate confirms this assessment. Promotion uses a verify-or-rollback procedure: the builder snapshots the library, places the lemma in its appropriate module (merging it with existing declarations when appropriate), and executes the integration chain—build, re-index, re-embed, lint, and regenerate documentation. Any failure restores the snapshot; only a clean integration is retained. Together, the two channels expand what

Table 5: The run catalogue: 123 recorded CausalSmith runs by disposition. A separate literature-reproduction track adds one reproduced partial-identification result. Counts are entries in the run record; a re-run that supersedes an earlier attempt at the same question counts separately, and ten legacy runs from the retired proposal track that predates the current pipeline are excluded.

Disposition	Runs	Gate that determined it
Accepted	9	sound, novel at tier, proved in Lean 4
Downgraded	44	sound, below the novelty target
Failed	70	rejected at the proposal or the mathematics
Total	123	

the pipeline knows (the run record) and what it can reuse (the library), and jointly reduce the burden of subsequent runs.

5.5 Presentation

The presentation stage consumes the reviewed graph. It converts an accepted result into a working paper in which each theorem, definition, and assumption links to its verified Lean 4 source. Linking prose to code is established practice [56]. The link records both facts attached to a graph node: the Lean 4 object compiles, and the statement has been marked `matched` to the English claim. An equivalence check applies the criterion of Section 5.3: the paper statement and the Lean 4 declaration must express the same proposition, with neither stronger nor weaker than the other. The check runs before the draft is revised. On failure, the pipeline stops for human adjudication and preserves the intended declaration target.

6 Results

Our evidence is what the system produced, drawn entirely from the pipeline's run records, logs, and library index. We give the distribution of outcomes across runs, report which kinds of self-proposed question the system converts into accepted results, examine one accepted result in depth, report the axiom audit that screens each accepted result, and trace the library feedback loop closing on a concrete lemma. We treat each as evidence for a claim in Section 1 and defer the more demanding experiments—precision and recall of the audit, cost accounting, and an independent significance panel—to future work. The pipeline also attaches a research-quality signal to each result: an adversarial novelty gate assigns a tier, and the presentation stage scores the finished write-up. These are LLM judgments of exactly the kind this paper argues is unreliable (Section 1), so we report them without treating them as a validated evaluation of significance; Section 7 discusses why validating them is itself an open problem.

6.1 The run catalogue

The run record holds 123 runs, distributed across three dispositions (Table 5). A run is accepted only when it is sound, novel at its requested tier, and proved to completion in Lean 4; nine runs meet this bar. The remaining runs are retained rather than discarded: as Section 5.4 describes, downgraded and failed runs feed the record that screens later proposals.

The nine accepted results span causal discovery, statistical and causal estimation, panel methods, and experimentation; identification and structural-causal-model results appear only in the lower tiers.

Table 6: The run catalogue by the cluster Discovery assigned at proposal time. "Accept rate" is accepted runs over total runs in the cluster. The rule dividing the two blocks is described in the text; it is our reading of where the difficulty of a run falls, not part of the clusters' definitions. Eight of the nine accepted results fall in the upper block.

Cluster	Acc.	Down.	Fail.	Total	Accept rate
`Stat`	4	7	4	15	27%
`Experimentation`	3	3	1	7	43%
`Panel`	1	0	6	7	14%
`ExactID`	1	8	27	36	3%
`PartialID`	0	24	30	54	0%
`SCM`	0	2	2	4	0%
Total	9	44	70	123	7%

The contrast with the size of the library (Section 4) is worth stating plainly: although Causalean holds thousands of machine-checked declarations, the number of fully accepted novel discoveries is small. The evidence therefore supports the proposed mechanism—kernel-checked proofs paired with graph-localized statement review—but assessing discovery performance across causal inference will require a larger evaluation.

6.2 Which self-proposed questions the system converts

The runs also carry a signal about what the topic selector is good at asking, not only about how often it succeeds. Discovery assigns every run to one of six clusters before any mathematics is attempted, so the assignment is fixed independently of the outcome. Sorting the catalogue by cluster (Table 6) leaves the accepted results concentrated in `Stat`, `Experimentation`, and `Panel`, and nearly absent from `ExactID`, `PartialID`, and `SCM`, where 94 runs produced a single acceptance. The rule the split follows is not the clusters' subject matter, which is broader than any one kind of result, but where in a run the difficulty falls. A run in the first group typically takes an estimand whose identification is settled and has to establish something analytical about it; a run in the second has to produce the identification argument itself. The first kind of question can be posed against a known quantity; the second cannot.

Reading the accepted topics themselves sharpens the pattern beyond the cluster labels. Each of the nine names a specific published result and closes a technical gap that the literature has already located; the targets include a matching anisotropic-Hölder converse to a published higher-order-influence-function upper rate; the $\log^2$ gap of Section 6.3; a differentially private counterpart of a known conditional average treatment effect rate; a two-sided welfare-regret rate under decaying overlap; a demonstration that a published semidefinite design's Gaussian rounding certificate is one-sided rather than exact; a matched Chebyshev lower bound for a rollout design; and, in the one accepted identification run, a proof that a published order-$(2m{+}3)$ cumulant test for causal direction is not minimal, the direction being generically recoverable one order lower. In each case the target was well posed before the run began, and success was checkable against a published quantity.

The unsuccessful runs fail differently, and the run record says how. The recorded downgrade reasons in the identification clusters repeatedly report a sound derivation whose content collapses into an existing construction: reviewers describe the delivered result as "generic Manski-style outer containment," "standard finite response-type sharpness," "shallow two-point mean-completion sharpness," or a "one-line bookkeeping transfer." These runs were not stopped by an unprovable

step or an unfinished formalization. They were stopped because the mathematics, once derived, turned out to be a known idea in new notation. The binding constraint in these clusters is conception rather than proof.

A plausible mechanism is that the two kinds of question differ in how well posed they are at proposal time. A technical gap supplies the proposer with a target and the gates with a criterion: the converse must match the published upper bound, the estimator must attain the stated rate, the test must use one fewer cumulant order. The pipeline can grade partial progress against that criterion, and the run either reaches it or visibly does not. A question whose contribution must be a new identifying idea or a new framing supplies no such target. The proposer can state a plausible-sounding object and pass the proposal gate, and only the derivation review discovers that the object reduces to something known—by which point the run has consumed its budget. This is consistent with the observation that the identification clusters produce many downgrades rather than many outright refutations.

We report this as an observation about the current system, not a claim about capability limits, and it is uncontrolled in three ways. The clusters differ in the topic mix the selector proposed and in the novelty tier requested, so the comparison is not between matched questions. CAUSALEAN's coverage is deeper in estimation and statistical theory than in partial identification (Table 2), so library support and question type are confounded; a run that needs a new bound construction also tends to need new substrate. And the accept/downgrade boundary is set by the pipeline's own novelty gate, an LLM judgment of exactly the kind Section 7 declines to treat as validated. Separating question type from library support—by proposing matched technical and conceptual questions within a single cluster—is the experiment this observation calls for and that we have not run.

6.3 A flagship result: closing a minimax gap for the ATE

The strongest accepted result closes a standing gap in the minimax theory of average-treatment-effect (ATE) estimation under high-dimensional discrete confounding. Zeng et al. [54] established the minimax lower scale $n^{-1} + \big(d/(n \log n)\big)^2$ for estimating the ATE from n i.i.d. observations (X, A, Y) with a discrete confounder $X \in \{1, \dots, d\}$, binary treatment A and outcome Y, and strict-interior overlap $\epsilon \le \Pr(A = 1 \mid X = k) \le 1 - \epsilon$. The estimators they analyze, however, leave a $\log^2$-factor gap between the known upper and lower bounds. The run constructs a single computable estimator that closes it.

Under the stated overlap, consistency, and conditional-exchangeability assumptions, the target is the adjustment functional $\tau(P) = \sum_{k=1}^{d} p_k(\mu_{1k} - \mu_{0k}) = \mathbb{E}[Y(1) - Y(0)]$, where $p_k = \Pr(X = k)$ and $\mu_{ak} = \mathbb{E}[Y \mid A = a, X = k]$. The estimator splits the sample in two, uses a pilot count to label each covariate cell heavy or light, and treats the two regimes differently: heavy (well-populated) cells receive a plug-in ratio estimator, while light (sparsely sampled) cells receive a best-polynomial approximation of the cell functional with unbiased factorial-moment lifting. A single universal numerical calibration—fixed constants that do not take the overlap ϵ as input—controls the split, the polynomial degree, and the variance normalization, and the estimator runs in $O(dM^4)$ arithmetic operations. The headline theorem states that, for every fixed $0 < \epsilon < 1/2$, this estimator attains

$$\mathsf{R}_{n,d,\epsilon} \asymp_{\epsilon} \frac{1}{n} + \Big(\frac{d}{n \log n}\Big)^2 \qquad \text{uniformly for } d \lesssim_{\epsilon} n \log n,$$

so the minimax MSE has parametric order n^{-1} whenever $d = O(\sqrt{n} \log n)$ and tends to zero if and only if $d = o(n \log n)$. It is formalized as `sharp_minimax_fixed_interior` across twenty-six Lean 4

modules, together with the light-cell approximation lemma, the heavy-cell aggregation bound, the universal-tuning corollary, and a verified two-category confounding witness.

The logic graph records the scope of the contribution and makes the boundary auditable. The novel, proved part is the upper bound: the hybrid estimator and its matched rate. The minimax lower half is not reproved here; it is transferred from the published moment-matching bound of Zeng et al. [54]. In the Lean 4 development this borrowed bound appears as an explicit hypothesis on the headline theorem—a cited node carrying its source—rather than an `axiom` inserted into the proof. This follows the borrowed-result discipline of Table 4: a load-bearing external result is carried as a typed hypothesis with source evidence, so a reader sees precisely which part is proved in this work and which is imported. The result was accepted at the field-novelty tier.

6.4 Machine-checked soundness

Each accepted result is screened for the unproved-shortcut failures of Table 4 before it is recorded. In a clean build, `#print axioms` is run on the headline theorem and cross-checked by a comment-aware scan of the recorded module and its reachable library closure for `sorry`, `admit`, `native_decide`, and `axiom` (Section G). For the flagship result, the twenty-six modules of the development contain none of these, and the headline theorem reduces to Lean 4's standard axioms; its single external input, the Zeng et al. [54] lower bound, is visible as a hypothesis, not an axiom. This is what separates a kernel-accepted proof that rests only on its stated assumptions from one that has quietly promoted an open step to an `axiom`.

6.5 Library feedback in action

The library feedback loop can be traced through a complete instance. Proving the converse half of a dose-response result from another accepted run required a Bretagnolle–Huber affinity bound for arbitrarily many hypotheses, absent from Mathlib. The library builder proved this result and promoted it to CAUSALEAN as `Causalean.Stat.bretagnolle_huber_affinity`; a subsequent result, the Le Cam two-point reduction on which the converse depends, now imports and reuses it. Thus a run required a lemma, the system proved it, promotion added it to the library, and a later result consumed it. Thirteen library builds have produced such promotions, including Fano's inequality, a KL density-tilt expansion, and Chebyshev design tools. Estimating how library growth trades off against cost per result remains future work (Section 7).

7 Discussion and Limitations

CAUSALFORGE separates three questions. First, the Lean 4 kernel checks that a proof is valid. Second, the statement audit checks node by node whether each formal statement expresses the intended claim; this review can still miss errors. Third, the importance of a final paper remains a human judgment, informed by the pipeline's novelty tiers but not determined by them. Keeping these questions separate prevents a valid proof from being mistaken for a meaningful or important result. The pipeline machinery is largely domain-independent, whereas the formal library is specific to causal inference. Applying the approach to another field requires building or adopting a comparable library and retrieval interface. The graph, audit, and library feedback loop should transfer with limited modification.

Our reporting has several limitations, the first of which concerns research quality. The pipeline estimates it: an adversarial novelty gate assigns tiers and the presentation stage scores each write-up. But these are LLM judgments, and the independent human evaluation that would validate them

is absent. We therefore report what the system produced and how the pipeline rated it, not an established measure of significance. Building such a measure is itself an open problem: unlike proof soundness, which the kernel decides, or statement match, which reduces to a local comparison, the value of a correct theorem resists the labeled ground truth a benchmark requires. We regard constructing such a benchmark, and convening an independent panel, as future work. Second, the faithfulness audit itself relies on an LLM. The task it poses is narrower than open-ended review, and frontier models performed well on it in our observations, but a model can still err when auditing a statement, and we do not estimate how often it does. Third, the cost and reuse evidence is qualitative because the runs record iteration depth but not token use or elapsed time, and the accepted catalogue is small and concentrated in estimation and experimentation.

8 Conclusion

Automated theoretical research requires a reliable basis for evaluating correctness, yet LLM reviewers can be misled. We have described a causal-inference system that uses formal verification to establish proof soundness while making the limits of its other assessments explicit. A verified library reduces formalization cost; a topic-selecting, graph-audited pipeline proposes and proves results while assessing whether each statement expresses its intended claim; and a library feedback stage expands the library with lemmas required by completed runs. The resulting trust model has three parts: the kernel checks the proof, the statement audit checks each formal statement against the intended claim, and significance remains a human judgment. The principal next steps are a rigorous study of audit reliability, controlled ablations against kernel-only and LLM-judge-only baselines and an independent novelty panel provied from expert judgement.

References

[1] Sawal Acharya, Terry Jingchen Zhang, others, and Zhijing Jin. CauSciBench: Can LLMs automate causal inference in real-world scientific research? *arXiv preprint / OpenReview*, 2025. End-to-end causal-inference benchmark over real-world scientific research; causalNLP group.

[2] Daneshvar Amrollahi, Jerry Lopez, and Clark Barrett. Faithful autoformalization via roundtrip verification and repair. *arXiv preprint arXiv:2604.25031*, 2026.

[3] Anonymous. Causal AI scientist: Towards end-to-end causal inference with large language models. OpenReview submission (under review), 2026.

[4] Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023.

[5] Joeran Beel, Min-Yen Kan, and Moritz Baumgart. Evaluating Sakana's AI scientist: Bold claims, mixed results, and a promising future? *arXiv preprint arXiv:2502.14297*, 2025.

[6] Rafael Cabral, Tuan Manh Do, Xuejun Yu, Wai Ming Tai, Zijin Feng, and Xin Shen. Proof-Flow: A dependency graph approach to faithful proof autoformalization. *arXiv preprint arXiv:2510.15981*, 2025.

[7] Sirui Chen, Bo Peng, Meiqi Chen, Ruiqi Wang, Mengying Xu, Xingyu Zeng, Rui Zhao, Shengjie Zhao, Yu Qiao, and Chaochao Lu. Causal evaluation of language models. *arXiv preprint arXiv:2405.00622*, 2024.

[8] Joanie Hayoun Chung, Sumin Lee, and Sungbin Lim. ORCA: ORchestrating causal agent. *arXiv preprint arXiv:2508.21304*, 2025. CHI EA 2026.

[9] Chengxiao Dai, Zhaokun Yan, and Zhanhui Lin. The signal-coverage matrix: Stratifying type and semantic errors in statement autoformalization. *arXiv preprint arXiv:2606.28013*, 2026.

[10] Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction (CADE 28)*, pages 625–635. Springer, 2021.

[11] Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. *arXiv preprint arXiv:2303.04910*, 2023.

[12] Jinglong Gao, Xiao Ding, Bing Qin, and Ting Liu. Is ChatGPT a good causal reasoner? a comprehensive evaluation. *Findings of the Association for Computational Linguistics: EMNLP*, 2023. arXiv:2305.07375.

[13] Nikhil Garg. EconCSLib: AI-assisted lean formalization for economics and computation research. *arXiv preprint arXiv:2606.13306*, 2026.

[14] Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Vivek Natarajan, et al. Towards an AI co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.

[15] Sukjin Han. Mining causality: AI-assisted search for instrumental variables. *arXiv preprint arXiv:2409.14202*, 2024.

[16] Thomas Hubert, Rishi Mehta, Laurent Sartran, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 2025. DOI: 10.1038/s41586-025-09833-y.

[17] Vasily Ilin and Brian Nugent. Sorries are not the hard part: An expert-review case study of a semi-autonomous formalization. *arXiv preprint arXiv:2606.13925*, 2026.

[18] Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations (ICLR)*, 2023.

[19] Fengqing Jiang, Yichen Feng, Yuetai Li, Luyao Niu, Basel Alomair, and Radha Poovendran. BadScientist: Can a research agent write convincing but unsound papers that fool LLM reviewers? *arXiv preprint arXiv:2510.18003*, 2025.

[20] Zhijing Jin, Yuen Chen, Felix Leeb, Luigi Gresele, Ojasv Kamal, Zhiheng Lyu, Kevin Blin, Fernando Gonzalez Adauto, Max Kleiman-Weiner, Mrinmaya Sachan, and Bernhard Schölkopf. CLadder: Assessing causal reasoning in language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[21] Zhijing Jin, Jiarui Liu, Zhiheng Lyu, Spencer Poff, Mrinmaya Sachan, Rada Mihalcea, Mona Diab, and Bernhard Schölkopf. Can large language models infer causation from correlation? In *International Conference on Learning Representations (ICLR)*, 2024.

[22] Nitish Joshi, Abulhair Saparov, Yixin Wang, and He He. LLMs are prone to fallacies in causal inference. *arXiv preprint arXiv:2406.12158*, 2024.

[23] Emre Kıcıman, Robert Ness, Amit Sharma, and Chenhao Tan. Causal reasoning and large language models: Opening a new frontier for causality. *Transactions on Machine Learning Research*, 2023. arXiv:2305.00050.

[24] Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurelien Rodriguez, and Timothee Lacroix. HyperTree proof search for neural theorem proving. *arXiv preprint arXiv:2205.11491*, 2022.

[25] Chenyi Li, Shengyang Xu, Chumin Sun, Li Zhou, and Zaiwen Wen. Formalization of optimality conditions for smooth constrained optimization problems. *arXiv preprint arXiv:2503.18821*, 2025. `https://github.com/optsuite/optlib`.

[26] Zenan Li, Yifan Wu, Zhaoyu Li, Xinming Wei, Xian Zhang, Fan Yang, and Xiaoxing Ma. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2410.20936.

[27] Xiao Liu, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. Are LLMs capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data. In *Findings of the Association for Computational Linguistics: ACL*, 2024. arXiv:2402.17644.

[28] Jianqiao Lu, Yingjia Wan, Yinya Huang, Jing Xiong, Zhengying Liu, and Zhijiang Guo. FormalAlign: Automated alignment evaluation for autoformalization. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.10135.

[29] Jing Ma. Causal inference with large language model: A survey. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5901–5913. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.327. URL `https://aclanthology.org/2025.findings-naacl.327/`.

[30] Theodore Meek, Siyuan Ge, Di Qiu Xiang, Simon Chess, and Vasily Ilin. Formalizing numerical analysis: An agent pipeline and quality audit beyond kernel acceptance. *arXiv preprint arXiv:2606.14000*, 2026. Verify author-name segmentation against the arXiv page.

[31] Noor Islam S. Mohammad and Tamim Sheikh. The faithfulness gap: Certifying semantic equivalence between natural-language and formal mathematical statements. *arXiv preprint arXiv:2606.16541*, 2026.

[32] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[33] Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. Reliable evaluation and benchmarks for statement autoformalization. *Findings of the Association for Computational Linguistics: EMNLP*, 2025. arXiv:2406.07222.

[34] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

[35] Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

[36] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[37] Ayush Sawarni, Jiyuan Tan, and Vasilis Syrgkanis. Causalreasoningbenchmark: A real-world benchmark for disentangled evaluation of causal identification and estimation, 2026.

[38] ChengAo Shen, Zhengzhang Chen, Dongsheng Luo, Dongkuan Xu, Haifeng Chen, and Jingchao Ni. Exploring multi-modal data with tool-augmented LLM agents for precise causal discovery. *arXiv preprint arXiv:2412.13667*, 2024.

[39] Ivaxi Sheth, Zhijing Jin, Bryan Wilder, Dominik Janzing, and Mario Fritz. IV co-scientist: Multi-agent LLM framework for causal instrumental variable discovery. *arXiv preprint arXiv:2602.07943*, 2026.

[40] Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean copilot: Large language models as copilots for theorem proving in lean. *arXiv preprint arXiv:2404.12534*, 2024.

[41] Sho Sonoda, Kazumi Kasaura, Yuma Mizuno, Kei Tsukamoto, and Naoto Onda. Lean formalization of generalization error bound by Rademacher complexity and Dudley's entropy integral. *arXiv preprint arXiv:2503.19605*, 2025. `https://github.com/auto-res/lean-rademacher`.

[42] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, pages 367–381, 2020.

[43] Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482, 2024. doi: 10.1038/s41586-023-06747-5.

[44] George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. PutnamBench: Evaluating neural theorem-provers on the Putnam mathematical competition. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2407.11214.

[45] Zeynel A. Uluşan, Burak S. Akbudak, Can S. Erer, and Gözde Gül Şahin. FormalRewardBench: A benchmark for formal theorem proving reward models. *arXiv preprint arXiv:2605.10141*, 2026.

[46] Xinyue Wang, Kun Zhou, Wenyi Wu, Har Simrat Singh, Fang Nan, Songyao Jin, Aryan Philip, Saloni Patnaik, Hou Zhu, Shivam Singh, Parjanya Prashant, Qian Shen, and Biwei Huang. Causal-Copilot: An autonomous causal analysis agent. *arXiv preprint arXiv:2504.13263*, 2025.

[47] Tingzhou Wei, Zeyu Zheng, Ethan X. Fang, and Junwei Lu. Hypothesis-disciplined multi-agent automated formalization of asymptotic statistical theory. *arXiv preprint arXiv:2606.20642*, 2026.

[48] Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[49] Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

[50] Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

[51] Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in AI. *arXiv preprint arXiv:2412.16075*, 2024.

[52] Linying Yang, Vik Shirvaikar, Oscar Clivio, and Fabian Falck. A critical review of causal reasoning benchmarks for large language models. *arXiv preprint arXiv:2407.08029*, 2024.

[53] Matej Zečević, Moritz Willig, Devendra Singh Dhami, and Kristian Kersting. Causal parrots: Large language models may talk causality but are not causal. *Transactions on Machine Learning Research*, 2023. arXiv:2308.13067.

[54] Zhenghao Zeng, Sivaraman Balakrishnan, Yanjun Han, and Edward H. Kennedy. Causal inference with high-dimensional discrete covariates. *arXiv preprint arXiv:2405.00118*, 2024.

[55] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. MiniF2F: A cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

[56] Thomas Zhu, Pietro Monticone, Jeremy Avigad, and Sean Welleck. LeanArchitect: Automating blueprint generation for humans and AI. *arXiv preprint arXiv:2601.22554*, 2026.

A Flagship theorem locations

For reproducibility, Table 7 gives the `Lean 4` declaration name and source location of each flagship result named in Section 4.3. All locations are relative to the `Causalean/` package root at the pinned toolchain `leanprover/lean4:v4.29.0-rc3`.

Table 7: Source locations of the flagship declarations in Table 3.

Declaration	File	Line
`markovEquiv_iff_sameSkeleton_sameImmoralities`	`Graph/MarkovEquiv.lean`	51
`do_rule2_kernel`	`SCM/Do/DoCalculus.lean`	97
`backdoor_identifiable_ae`	`SCM/ID/Backdoor.lean`	377
`frontdoor_identifiable_ae`	`SCM/ID/Frontdoor.lean`	1190
`id_sound_discrete`	`SCM/ID/GraphicalThms/IDSoundDiscrete.lean`	30
`lingam_identifiability_kurtosis`	`Discovery/LiNGAM/LiNGAMKurtosis.lean`	69

Table 7 continued

Declaration	File	Line
icp_sound	Discovery/InvariantPrediction/Soundness.lean	34
icp_complete_linearGaussian	Discovery/InvariantPrediction/.../Completeness.lean	270
disentanglement_identifiability	Discovery/LinearDisentanglement/Identifiability.lean	35
late_wald	PO/ID/Exact/LATE.lean	413
att_did	PO/ID/Exact/DID.lean	150
att_csdid	PO/ID/Exact/CSDID.lean	423
rdd_identification	PO/ID/Exact/RDD/SharpRDD.lean	295
frd_identification	PO/ID/Exact/RDD/FuzzyRDD.lean	575
whenToTreat_wald	PO/ID/Exact/DynamicLATE/WhenToTreat.lean	332
manski_bounds_ATE	PO/ID/Partial/Manski/NonAsp.lean	72
balkePearl_sharp	PO/ID/Partial/BalkePearl/Sharp.lean	852
lee_bounds_ATT_AS	PO/ID/Partial/Lee/Main.lean	33
imbensManski_pointwise_coverage	PO/ID/Partial/Inference/ImbensManski.lean	171
dml_ATE_tendstoNormal	Estimation/ATE/DML.lean	899
dml_ATE_attains_hahn_bound	Estimation/Efficiency/ATEVariance.lean	415
dml_ATT_isAsymLinear	Estimation/ATT/DML.lean	237
plr_dml_tendstoNormal	Estimation/PLR/DML.lean	158
seqDR_dml_isAsymLinear	Estimation/DTR/DTRInstance.lean	145
minimax_lower_bound_var_causal	Estimation/MinimaxATE/Causal/Minimax.lean	172
gmm_efficiency	Stat/GMM/VarianceAlgebra.lean	106
uStatisticOrder_clt	Stat/UStatistic/OrderM/CLT.lean	52
twfe_po_decomposition	Panel/.../CausalDecomposition.lean	81
bjs_linear_unbiased_iff_imputation_form	Panel/.../ImputationEventStudy/PanelBridge.lean	370
apparent_pretrends_from_post_treatment_of_cellGrid	Panel/.../EventStudyContamination/Contamination.lean	63
htEst_consistent_eate	Experimentation/UnknownInterference/Consistency.lean	133
directEffect_clt	Experimentation/.../Asymptotic/CLT.lean	118
localDependenceCLT_of_stein	Experimentation/.../SteinInstance.lean	176
wald_coverage_of_stein	Experimentation/.../SteinInstance.lean	252

B Logic-graph schema

The logic graph of Section 5.1 is persisted as a `FormalizationGraph`, one per result. A node records its kind (one of `setup`, `definition`, `assumption`, `lemma`, `theorem`, `gate`), its natural-language statement and Lean 4 declaration, a review status (`unreviewed`, `matched`, `derived`, `drift`), and a gate class (gated or cited). An assumption node carries one further label, distinguishing claim refinement from regularity bookkeeping and from a library gap. An edge records its kind (`statement-uses`, `proof-uses`, `setup-of`) and its endpoints. A schema validator rejects duplicate nodes and edges and reports structural-invariant violations.

C Operational specification of the pipeline

This appendix specifies the pipeline of Section 5 at the level of the implementation used for the reported runs. A theorem run is identified by a question identifier and a specialization, and the

orchestrator advances it through the ordered stages of Figure 4 and table 8. These numbered stages refine the four stages named in Section 5: D–1.1 through D0.5 carry out Discovery; F1, F1.5, and F2 carry out Formalization; F3 is proof construction; F2.5, F3.5, and F4 are the statement audit of Section 5.3 applied at three points of the proof loop; and F5 prepares the completed record, which the separate presentation state machine of Section F then consumes.

The orchestrator writes a stage's completion marker only after the stage returns, so a stopped run resumes from its recorded state, and a stage may return a checkpoint or an escalation instead of an advance. Autonomy is bounded in a specific way: the orchestrator executes the specified routine on its own and intermediate messages stay inside the run record, while the designated scientific and final-approval decisions remain operator-controlled.

Topic selection belongs to Discovery. A run invoked with no topic dispatches the `causalsmith-topics` module before the numbered D-stages: the selector searches and closely reads recent work, screens a candidate slate against active and recorded runs, and requires an adversarial topic-gate acceptance before it returns the topic, question identifier, and specialization. A researcher-supplied topic bypasses the selector and enters the same Discovery stage.

Table 8: Operational stages of a CausalSmith theorem run. The internal state uses the numerical identifiers; the D/F prefixes distinguish discovery from formalization in the command-line interface and logs.

Stage	Purpose	Action and acceptance condition	Principal durable output
D–1.1	Problem reconnaissance	Searches open problems and prior proposals for usable gaps before a proposal is written.	Gap record
D–1.2	Proposal construction	Produces a typed proposal and its mathematical core. The core identifies the objects, atomic assumptions, statements, and their dependency structure.	Proposal and typed core
D–0.5	Proposal gate	Reviews the proposal for novelty, duplication, and a viable mathematical direction. A rejection or a repairable finding returns the proposal to the appropriate discovery work before formalization begins.	Review decisions and revision record
D0	Mathematical derivation	Derives the proposed result and renders the research note. The solver may narrow an overstrong claim when the derivation warrants it, while preserving the scientific contract for later Lean 4 proof.	Derivation note and updated core

Table 8 continued

Stage	Purpose	Action and acceptance condition	Principal durable output
D0-max	Maximality checkpoint	After a clean discharge, halts the run and consults an external reviewer on the whole-paper maximization question: is there a sharper bound, better construction, stronger reframing, or tighter constant available? A concrete improvement returns to D0 as a directive; a reviewer confirmation of maximality resumes into D0.5. The default is to improve; a weaker tier is chosen only under a reviewer-confirmed maximality decision.	Maximality decision (verbatim reviewer finding logged)
D0.5	Derivation gate	Separates a fresh mathematical rederivation from the structural, novelty, and tier decision. A further cold review is used when the target tier requires it.	Mathematical, structural, and tier reviews
ckpt D/F	D0.5→F1 go/no-go	Run halt after D0.5 passes. The orchestrator decides whether the maximized, novelty-cleared result warrants committing to the expensive F1–F5 formalization; a resume enters F1, and a stop records or downgrades the discovery-only result.	Commit decision and lease re-grant
F1	Formalization plan	Maps every core node and the required ambient setup to a planned `Lean 4` object. For each reusable result, the plan records the intended declaration and module; for an unavailable fact, it records a disclosed gate that must be discharged.	`plan.json` and logic graph
F1.5	Plan and reuse gate	Runs deterministic checks for coverage, kind, dependency closure, declaration existence, and module placement, followed by a review of reuse fit and abstraction level. A clean plan can stop at the first checkpoint for an optional operator audit of depth, reuse, and statement fidelity; in automatic mode it auto-resumes.	Plan-gate and reuse reviews

Table 8 continued

Stage	Purpose	Action and acceptance condition	Principal durable output
F2	Lean 4 scaffold	Emits or revises a Lean 4 scaffold. The first scaffold may contain proof obligations; a subsequent revision edits only the flagged declarations and their necessary dependents, preserving already valid proof bodies.	Tagged Lean 4 source tree
F2.5–F4	Proof and statement review loop	First checks that the scaffold realizes the frozen specification (F2.5); then runs the proof-review loop that fills obligations and rechecks the changed frontier against the live Lean 4 compiler (F3); then runs the deterministic unused-hypothesis lint and scan for forbidden proof steps (F3.5); finally performs a full dual-model convergence review of the frozen graph (F4). The compatibility stage numbers F2.5, F3, F3.5, and F4 label these substeps; the loop owns their work.	Proof reviews, graph verdicts, crosswalk, and gate ledger
F5	Final approval preparation	Proceeds only after scans for forbidden proof steps and correspondence checks are clean, emits a final lemma-inclusive TEX–Lean 4 crosswalk, updates the API documentation, and stops at the second checkpoint. Recording, committing, and promotion require an explicit human decision.	Complete crosswalk and API update

The split between D0 and F1 carries most of the weight in this ordering. D0 fixes the mathematical contract: its typed core specifies what is to be formalized. F1 converts that core into a one-to-one implementation plan, so the intended use of library results, new lemmas, definitions, and assumptions is inspectable before proof search begins. Because the two are separate artifacts, a reviewer can locate a defect in the mathematics, in the plan's reuse decision, in the Lean 4 statement, or in the proof, rather than in the run as a whole.

D Run record, artifacts, and recovery

Every run has a durable directory of its own, partitioned by run kind and question identifier: a theorem run lives under `CausalSmith/doc/research/active/` until it is banked, and a study run under `CausalSmith/doc/study/`. The directory root holds the authoritative `state.json`, the append-only `pipeline.jsonl`, and the graph; discovery and formalization artifacts occupy separate subdirectories, and `reviews/` holds the review event log and the individual referee reports. The state records the last completed stage, the next action, flags that block unsafe continuation, the

selected novelty target, and the locations of the run's artifacts. State updates write a temporary file and rename it, so recovery always sees either the previous complete state or the next complete state, never a partial one. The pipeline log records each stage, its status, duration, message, and any next-step guidance.

The record separates durable evidence from transient diagnostics. The durable evidence is the proposal, derivation note, typed core, formalization plan, graph, Lean 4 files, review reports, and TEX–Lean 4 crosswalk. The transient logs—per-stage agent transcripts, reviewer debugging output, and liveness information—are kept under `logs/`. The separation keeps the run directory readable at its root while preserving enough detail to reproduce a decision or diagnose a failed stage.

Recovery is state-based. A resumed run loads and validates `state.json`, reconstructs the next stage from the ordered stage list, and reuses the artifacts that precede that stage. Explicit re-entry may re-run an earlier stage after a correction; the corrected stage then runs forward through its dependent gates. The orchestrator also recognizes bounded nonconvergence, and stops with a named route rather than looping: revise the plan, repair the source claim, build a missing library lemma, record a partial result, or abandon the run. A run that has reached F5 resumes as complete, and an explicit re-entry records the operator's decision to re-audit.

E Graph-controlled formalization, proof construction, and audit

The graph is the control surface connecting the mathematical core, the prose specification, the Lean 4 source, and the audit. F1 constructs it from the typed core and the formalization plan. F2 adds stable source annotations and extracts the emitted declarations, proof states, and dependencies back into it. The graph therefore carries two maps at once: a forward implementation map, from each planned object to what it should become, and a reverse audit map, from each emitted declaration to what it realizes.

Each review status carries a deliberately narrow meaning. `unreviewed` marks a node whose statement match has not yet been assessed; `matched` means the audit accepted the correspondence for the node's current content; `derived` identifies a pipeline-introduced intermediate object; and `drift` identifies a mismatch that must be repaired or adjudicated. Every accepted review is stamped with a content fingerprint. When a node or its realization changes, the fingerprint changes, and that node together with the affected dependency frontier returns to review. The final convergence pass deliberately forgoes this optimization and revisits the full frozen graph.

The proof loop runs in two phases, and the statement phase comes first. In it, the reviewer compares the frozen natural-language statement and definitions against the Lean 4 scaffold. A mechanical scaffold mismatch returns to F2 for a localized revision. A false claim, a wrong or under-specified frozen statement, or a genuinely missing library fact is escalated instead, because rewriting the scientific contract to make the checker pass would defeat the purpose of the audit. The natural-language content of a frozen, from-note node is guarded throughout the loop, so the system repairs a divergent Lean 4 realization while preserving the statement that realization is meant to express.

Proof filling begins only once the statement phase is clean. Each iteration reviews the modified frontier, attempts the remaining proof obligations against the live compiler and the retrieved library results, refreshes the graph from the resulting source tree, and tests whether the graph has made progress. Completion requires a successful build; a research tree free of real `sorry` and `admit`; frozen theorems and their relevant dependency closures both proved and matched; and a clean result from the deterministic unused-hypothesis lint. The loop also scans comment-stripped source for the tokens that mark forbidden proof steps, including `axiom`, `opaque`, `native_decide`, and `unsafe`.

Completion thus depends on proof closure and on clean proof-checking evidence, not on either alone.

The final convergence review is independent of the incremental frontier review. It re-audits every frozen statement and records the treatment of every library dependency. A gated dependency appears as an explicit conditional assumption or as a build obligation with a visible status. A cited dependency is matched against its cited source, and blocks final approval if the encoding is mismatched or underspecified. Only after this review does F5 emit the complete crosswalk, checking that every row that claims an exact or equivalent correspondence carries a Lean 4 anchor. The crosswalk is the final inspectable ledger, and its entries move with the proof revisions behind them.

F Presentation pipeline and library feedback

The theorem pipeline and the paper pipeline are separate state machines. The paper pipeline consumes a recorded result and runs stages P0–P5. P0 prepares literature evidence; P1 creates an outline and runs a statement audit against the frozen graph; P2 drafts the paper and audits proof-related prose; P3 executes the remaining presentation gates; P4 emits the rendered, linked artifact; and P5 obtains a final referee review and routes its findings to the earliest stage that can address them. P1 and P2 are deliberate outline and draft checkpoints: resuming from either approves continuation, while explicit re-entry at P0, P1, or P2 permits a targeted revision followed by the downstream gates.

The paper-to-code link is assembled graph-first. For each displayed object, the presentation system takes the relevant declaration and its `statement-uses` neighbors from the verified graph, emits a source anchor, and checks that the displayed dependencies and assumptions have corresponding links. This is what separates a link that merely points at compiling code from a link whose object has passed the statement audit. The English statement and its Lean 4 counterpart must be logically equivalent in both assumptions and conclusion; if either is stronger than the other, the pipeline halts for adjudication and preserves the intended declaration target.

A missing reusable lemma is treated as an explicit library feedback event. The theorem run identifies it as a library gap and routes it to study mode, which scaffolds and proves the lemma in the library, verifies its integration, refreshes the retrieval artifacts and documentation, and returns the resulting declaration to F1 as a candidate for reuse. Promotion is deliberately not an automatic side effect of solving one paper: the integration and the library-facing claim it creates remain subject to the human recording and promotion decision. This is how CausaLean grows from pipeline demand while keeping an auditable boundary between a run-specific result and a reusable library theorem.

G Reproducibility

The library and pipeline are pinned to Lean 4 toolchain `leanprover/lean4:v4.29.0-rc3`. The declaration counts in Table 2 are read from the compiled environment index (`lake exe library_index`); the axiom-check results in Section 6.4 are produced by `#print axioms` on each headline theorem run in a clean build, cross-checked by a comment-aware scan for `sorry`, `admit`, `native_decide`, and `axiom` over each recorded module and the reachable library closure it imports.

Source, library, and run record are at `https://github.com/Jiyuan-Tan/CausalForge`, with a browsable view of the library and of the accepted results at `https://jiyuan-tan.github.io/CausalForge/`. The run catalogue of Tables 5 and 6 is the set of banked run directories under `CausalSmith/doc/research/_bank/`, one directory per run, filed under `accepted/`, `downgraded/`, or `failed/`; each carries the verbatim run artifacts (state file, proposal, review log, derivation

note) that the dispositions and the quoted downgrade reasons are read from. Legacy runs—those predating the current pipeline, identified by a `q`-prefixed identifier and a null cluster field—are excluded from both tables. The cluster labels in Table 6 are CausalSmith's own six clusters, recorded in each run's state file and encoded in its identifier prefix (`stat_`, `exp_`, `panel_`, `eid_`, `pid_`, `scm_`); Discovery assigns the cluster at proposal time and selects the cluster-specific setup prompt from it, so the grouping is fixed before any of the run's mathematics or its outcome is known.

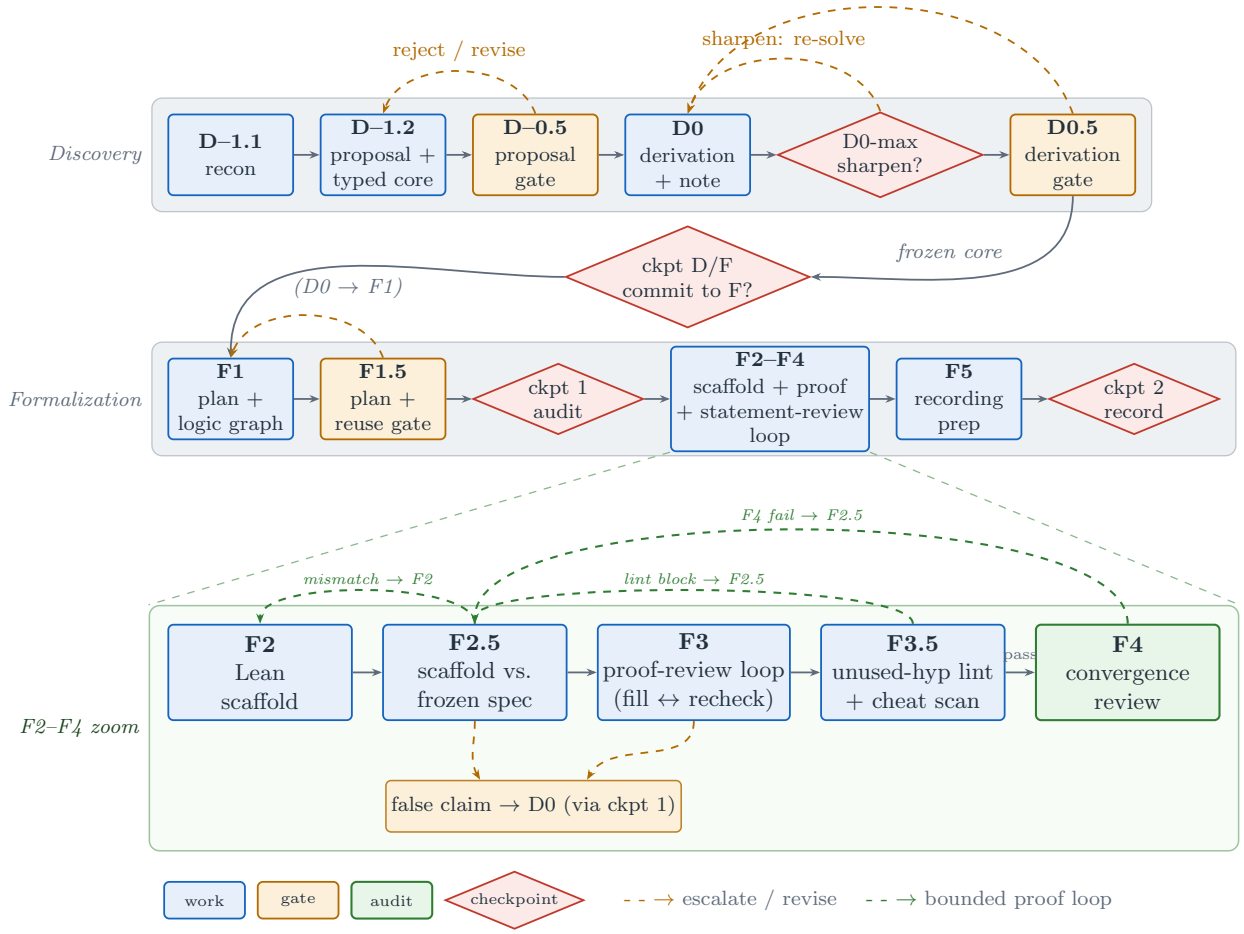

Figure 4: Operational stage flow of a CausalSmith theorem run, with an inset zoom into the F2–F4 proof and statement-review loop. Top: Discovery (top band) produces a proposal and typed mathematical core, which is frozen at D0.5 and handed to Formalization (middle band). Gates (amber) may reject or return work to the preceding stage; four red diamonds mark run halts for external judgment: the D0-maximality checkpoint (D0-max) after a clean derivation asks an oracle whether the result can be sharpened before D0.5 freezes it; the D/F go/no-go (ckpt D/F) is a run halt after D0.5 where the orchestrator decides whether to commit to the expensive F1–F5 phase; and two operator-optional checkpoints can request a human decision before F2 (ckpt 1) and before final recording (ckpt 2). In the automatic mode used for most runs, ckpt 1 auto-resumes and only the final acceptance at ckpt 2 is normally held for a person. Bottom (zoom panel, connected by dashed lens lines): the five substages inside the proof loop, F2 through F4. **F2** emits or revises the Lean 4 scaffold. **F2.5** compares that scaffold against the frozen specification: a mechanical mismatch returns to F2 (green mismatch → F2 arc), whereas a false or under-specified claim escalates to D0 (amber → D0) rather than silently rewriting the contract. **F3** is the proof-review loop that fills the remaining proof obligations against the live Lean 4 compiler and the retrieval interface, internally iterating fill ↔ recheck until the frontier is empty; a witnessed false claim escalates to D0 through the amber branch. **F3.5** runs the deterministic unused-hypothesis lint together with the comment-aware cheat-token scan (`sorry`, `admit`, `axiom`, `opaque`, `native_decide`, `unsafe`); a blocking lint finding re-enters at F2.5. **F4** is an independent dual-model convergence review of the full frozen graph; a fail returns to F2.5 for repair, and a pass advances to F5.