# CAUSALSMITH: A FORMALLY GROUNDED, SELF-IMPROVING AGENTIC FRAMEWORK FOR AUTOMATED RESEARCH IN CAUSAL INFERENCE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Automating theoretical research requires generating candidate results and evaluating them reliably. Models keep getting better at the first, while the second remains hard. A common approach asks one large language model (LLM) to review what another produced, yet such reviewers are empirically unreliable: they may accept fabricated papers and catch the fabrication at close to chance rates (Jiang et al., 2025). We present CAUSALSMITH, a framework for automated theoretical research in causal inference built on the Lean proof assistant, where a proof is checked by a program rather than read by a referee. It rests on CAUSALEAN, a foundational Lean library for causal inference holding 10,575 machine-checked definitions and theorems, developed with language-model assistance under human design and review. Around it we build a self-improving agentic pipeline that selects research topics, proposes results, formalizes statements, constructs proofs, and presents the resulting artifacts for human inspection, pairing Lean verification with a statement audit that compares each formal theorem against the informal claim behind it. We evaluate the system using artifacts produced by completed autonomous research runs. The source code, formal library, and run history are in the anonymous replication package submitted with this paper.

## 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. Empirical evaluations already show the defects of such reviewers. An independent evaluation of the original AI Scientist (Lu et al., 2024) reports coding failures, limited novelty, and methodological defects (Beel et al., 2025), and Jiang et al. (2025) show that LLM reviewers accept deliberately fabricated papers at rates reaching $82\%$ in their tested configurations, while their detection interventions perform near chance. These findings make LLM review an unreliable evaluator of correctness.

Formal verification offers a different basis for evaluation. A theorem written in Lean 4 is checked by a program rather than read by a referee. If Lean 4 accepts the proof, then every step of the argument has been verified against the axioms, so a gap or a mistake in the reasoning cannot survive. Recent work in formal mathematics shows that this does more than screen out errors—it extends what the models can reach. Pairing a model that proposes proof strategies with Lean 4 as the arbiter of each step has begun to settle open research problems rather than competition exercises: one such agent autonomously resolved nine open Erdős problems and forty-four OEIS conjectures in a single evaluation (Tsoukalas et al., 2026), building on earlier Lean 4-based systems at olympiad level (Hubert et al., 2026). Grounding a system in Lean 4 takes the correctness of proofs out of a model's hands entirely. However, two obstacles stand between that guarantee and an automated system that can use it: cost and faithfulness.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

The first obstacle is cost. Formalizing a result from first principles is slow work, so a system that proves causal-inference theorems needs a library to start from: identification theorems, estimators, and asymptotic results already stated and proved in the literature. The successes above are concentrated in combinatorics, number theory, and optimization, where Mathlib is deep enough that a new statement can be written down and attacked directly (The mathlib Community, 2020; Tsoukalas et al., 2026); causal inference has no comparable foundation. Without one, every run rebuilds standard theory before it can reach anything new.

The second obstacle is faithfulness. Lean 4 checks the proof, but it says nothing about whether the theorem is the one the system meant to state, so the natural-language statement may not match the theorem actually proved. Recent work documents this gap from several angles (Ilin & Nugent, 2026; Dai et al., 2026; Meek et al., 2026), and we saw it in our own runs: in one, an agent turned the hard lemmas into `axioms` and produced Lean 4 code that compiled with axioms standing in for proofs; in another, the model altered a hypothesis, so the Lean 4 code proved a weaker statement than the paper claimed. Both compile, and neither is a proof of what was claimed. Our pipeline guards against this with a fine-grained audit step (Section 5.2) that compares each formal statement with the claim it is meant to express. Lean 4 checks the proofs the pipeline produces, and the audit checks that the statements they establish are the ones the paper reports.

We instantiate this approach in causal inference through CAUSALSMITH, which has two components. The first is CAUSALEAN, a Lean 4 library covering the causal-inference toolkit, which gives agents verified pieces to build on. It was itself built with LLM assistance: humans chose what to formalize and how to state it, agents drafted the definitions, proofs, and docstrings, and a human then reviewed the resulting Lean 4 statements. The second is a self-improving agentic pipeline that selects or accepts a research topic, proposes a result, formalizes it, proves it, and presents it. It audits one statement at a time rather than judging a finished paper once, and promotes reusable lemmas into CAUSALEAN as they are proved and reviewed.

**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 $10{,}575$ machine-checked declarations, written with language-model assistance under human design, and provides a retrieval interface for agents (Section 4).
- An end-to-end research pipeline whose Discovery stage can select its own research topic and propose a causal-inference result, which the pipeline then formalizes, proves, and presents, together with a library feedback loop that adds lemmas proved during a run back into CAUSALEAN (Sections 5 and 5.2).
- An evaluation drawn from a $407$-run campaign: a catalog of machine-checked results and a headline result found by the system, which closes a gap in Zeng et al. (2024) (Section 6).

**CAUSALSMITH pipeline.** The pipeline has four stages, shown in Figure 1. Discovery takes a topic from a researcher or proposes one itself, then solves it in natural language under an adversarial review of novelty and soundness, storing the solution as a dependency graph over the individual statements of the paper. Formalization gives every node a Lean 4 declaration, reusing what CAUSALEAN already provides, and the statement audit checks node by node that each declaration says what its node claims. Proof construction fills those declarations against the compiler and promotes any reusable lemma the library lacked back into CAUSALEAN. Presentation writes the paper from the finished graph, and the run—accepted, downgraded, or failed—enters the run history that screens the next proposal. Section 5 describes the stages in full.

**Availability.** The anonymous replication package submitted with this paper contains the CAUSALEAN library, the research pipeline, and the run history backing Section 6 (Section J). A companion site, whose link is withheld for review, gives a browsable view of the library and of each accepted result, with a proof map, a link from every statement to the Lean 4 declaration it was audited against, and a slide deck; readers can verify, rate, and comment on individual statements (Section I).

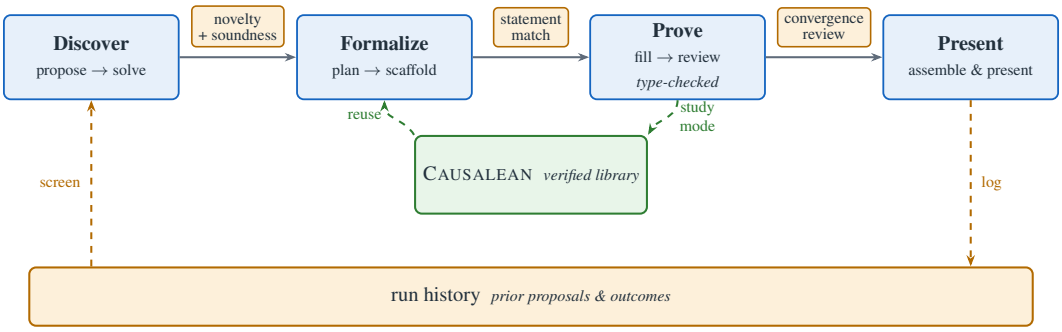



Figure 1: The CAUSALSMITH pipeline. A claim moves left to right through discovery, formalization, proof, and presentation, clearing a review between stages: novelty and mathematical soundness, statement matching, and a final dual-model convergence review of the frozen graph. Two feedback channels grow the system (Section 5.3): study mode proves a load-bearing lemma CAUSALEAN lacks and promotes it back into the library, and every run—accepted, downgraded, or failed—enters the run history that screens the next proposal. Sections E to H give the operational detail.

## 2 RELATED WORK

We discuss four lines of work: language models paired with proof assistants, and whether the statements they produce mean what was intended; agentic systems for automated research, and the verifiers their output is judged by; language models applied to causal inference; and formal libraries for statistics and economics. Section A gives the discussion at full length, with a table (Table 3) summarizing where CAUSALSMITH sits among the systems closest to it: it is the only one of them that proposes the theorem, machine-checks the proof, audits the formal statement against the intended claim, and feeds its own results back into the library it draws on.

**Autoformalization, theorem proving, and statement matching.** Language models have been paired with proof assistants for neural proof search (Polu & Sutskever, 2020; Lample et al., 2022), autoformalization (Wu et al., 2022; Jiang et al., 2023), whole-proof generation and repair (First et al., 2023), retrieval-augmented proving (Yang et al., 2023), and competition mathematics (Trinh et al., 2024; Hubert et al., 2026). Closest to us is autoformalization, where a model turns an informal statement into a formal one, and where evaluation relies on expert inspection (Wu et al., 2022), back-translation and semantic consistency checks (Li et al., 2024), probe-based faithfulness checks (Mohammad & Sheikh, 2026), and learned alignment scorers (Lu et al., 2025). In our setting the theorem is new, so the run writes its own informal claim and then formalizes it, leaving both sides of the comparison model output. Studies of this gap report formalizations with no remaining proof gaps that still fail expert review (Ilin & Nugent, 2026), and statements a proof assistant accepts that are semantically wrong (Dai et al., 2026; Meek et al., 2026). Our audit therefore back-translates, inside the discovery loop and localized over a proof-dependency graph, in the style of Lean blueprints and proof-flow tools (Zhu et al., 2026; Cabral et al., 2025).

**Automated research agents and verified discovery.** Agents that carry out end-to-end research (Yamada et al., 2025) are vulnerable when an LLM judges their output, whether shown by independent re-evaluation (Beel et al., 2025) or by adversarial construction (Jiang et al., 2025). Other work grounds discovery in a hard verifier: program search against a fitness function (Romera-Paredes et al., 2024; Novikov et al., 2025), or a proof against a type checker. Those fitness functions are fixed in advance, so a statement audit is a step they never need. "Co-scientist" agents fall in between: they propose and refine scientific claims but validate them empirically (Gottweis et al., 2026). The system closest to ours is a multi-agent pipeline for asymptotic statistics, whose auditor and reviewer agents check for unsupported strengthening of hypotheses (Wei et al., 2026). It formalizes known theory, whereas CAUSALSMITH proposes the theorem as well, and CAUSALSMITH localizes the audit over the dependency graph, so a changed statement re-opens the affected nodes rather than the whole formalization.

**LLMs for causal inference, and formal libraries.** A parallel literature studies whether language models reason causally, treating causal reasoning as something to test or deploy (Kıcıman et al., 2024; Ma, 2025): benchmarks probe graphical and correlational reasoning (Jin et al., 2023; 2024) and end-to-end causal inference on real studies (Acharya et al., 2026b; Sawarni et al., 2026), while a critical strand reports that surface accuracy conceals shallow reasoning (Gao et al., 2023; Joshi et al., 2024; Zečević et al., 2023). A third strand builds agents that do causal analysis without producing machine-checked causal theorems (Wang et al., 2025; Acharya et al., 2026a; Sheth et al., 2026); these systems evaluate outputs statistically or empirically. Separately, a Lean 4 ecosystem for theoretical statistics and economics has recently emerged (Lin et al., 2026; StatsMLlib contributors, 2026; Zhang et al., 2026; Garg, 2026), and CAUSALEAN overlaps with it wherever causal estimation theory needs the same tools. Those libraries formalize classical theory as the end product, whereas CAUSALEAN's statistical clusters exist to support theorems about causal estimands, and its core—structural causal models, do-calculus, graphical and partial identification, panel estimand characterizations, and design-based inference under interference—has no counterpart in any of them. CAUSALEAN is also built to be extended: each accepted run can promote its reusable lemmas back into the library for later runs to use.

## 3 BACKGROUND

**Causal inference.** Causal inference concerns the consequences of intervention rather than observation alone. Two frameworks are widely used. A structural causal model (SCM) represents causal relationships with a directed acyclic graph and interventions with the $\mathrm{do}$-operator; its classical results include the graphical identification criteria of 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 partially identified when the data determine only a set containing it, 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 a proof assistant and programming language in which mathematics is written out in complete formal detail (de Moura & Ullrich, 2021). A proposition is a type and a proof of it is a value of that type, so checking a proof means checking that a value has the type it claims; this check is called type-checking. The unit being checked is a declaration, a single named definition or theorem. Mathlib, Lean 4's community mathematics library, provides the analysis, probability, and measure theory that our library builds on (The mathlib Community, 2020).

This makes Lean 4 useful as an evaluator, because the check is mechanical and indifferent to how the proof is written. When a proof of a statement $\tau$ passes, $\tau$ follows from its stated assumptions together with the axioms the proof invokes, and Lean 4 reports those axioms on request. We call this property proof soundness, but it says nothing about whether $\tau$ is the proposition the researcher intended to prove, which Section 5.2 takes up.

Three things pass the type check while leaving the mathematics incomplete. Writing `sorry` in place of a proof tells Lean 4 to grant the statement, the formal counterpart of a `TODO`; declaring an `axiom` asserts a statement without proof; and a statement can be trivial, since a definition that unfolds to `True` is satisfied by everything. Lean 4 reports the first two on request—`sorry` raises a warning, and `#print axioms` lists what a theorem ultimately rests on—while the third is invisible to Lean 4 and has to be caught by reading the statement.

## 4 THE CAUSALEAN LIBRARY

The cost of formalizing results from first principles limits automated discovery. CAUSALEAN is a Lean 4 library for causal inference whose verified results agents can reuse and compose across runs.

### 4.1 DESIGN AND SCOPE

The CAUSALEAN library was first built with LLM assistance, following the division of labor of Section 1: humans set the scope, design the structure, choose which results to formalize, and fix the definitions that later theorems are stated against, while agents draft the statements, proofs, and docstrings. Nothing enters the library until it passes the Lean 4 type check and clears the axiom check (Section 4.2), and a human then reviews that the formal statement matches the intended claim and is stated in a standard form.

The compiled environment index lists $10{,}575$ declarations: $6{,}947$ theorems, $3{,}083$ definitions, and $545$ structures, instances, and inductive types, across $1{,}269$ files and roughly $362{,}000$ lines; Table 5 breaks this down by cluster.[1] The declarations are grouped into ten clusters, from graphical foundations to asymptotic statistics, layered so that the graphical and measure-theoretic foundations support the model languages, which support identification, which supports the estimation and design methods on top. Section C draws that structure (Figure 2) and walks the clusters in turn. The library also formalizes classical results from the literature. Section B catalogs the main result families across its major research areas: $34$ named declarations, each with a machine-checked proof and its source location.

### 4.2 RETRIEVAL AND AXIOM CHECKS

A library of this size is only usable if its declarations can be found, so it carries a keyword entry for every declaration: name, kind, module, source text, docstring, cross-references, axiom dependencies, and whether the proof uses `sorry`. The human-readable API documentation is generated from that index. On top of it we build a search engine for agents, driven in part by a retrieval encoder trained on CAUSALEAN itself, which ranks by keyword score and embedding similarity together and answers concept, type-pattern, and goal-directed queries (Section C).

The whole library is `sorry`-free, and none of the $10{,}575$ declarations rests on a hand-written `axiom`. The only non-standard axioms come from `native_decide`, a tactic that settles a finite computation by running the compiled code and trusting the result; these occur in finite-graph decidability arguments and several minimax calculations, and we report them explicitly.

## 5 THE CAUSALSMITH PIPELINE

CAUSALSMITH is a research pipeline with four stages: Discovery, Formalization, Proof Construction, and Presentation. It either takes a topic from a researcher or selects one itself, then proposes a causal-inference result in natural language, formalizes it, and produces a paper linked to the Lean 4 statements. A result is stored throughout as a dependency graph: Discovery creates it, formalization maps its nodes to planned Lean 4 declarations, proof construction fills those declarations, statement matching checks each formal node against the intended claim, and presentation writes the paper from the reviewed graph. An orchestrator runs the stages in a fixed order, retries a failed stage within fixed limits, and logs the verdicts. Figure 1 summarizes the workflow, and Sections E to H give the implementation details.

Lean blueprints and autoformalization pipelines already represent a development as a dependency graph of statements, each node carrying a formalization status (Zhu et al., 2026; Cabral et al., 2025). We adopt that structure and extend the node. Alongside its Lean 4 declaration, a node carries the intended natural-language statement and a review status saying whether the two have been compared and whether they matched, together with a class that separates two kinds of dependency: a must-prove node is a result the paper proves, and a cited node is one taken from the literature, carried with its source and lying outside the paper's novelty. The graph is therefore both a plan and an audit record, and when a statement changes the pipeline marks the downstream nodes unreviewed again and rechecks them. A typical completed result has on the order of a hundred nodes and a few hundred edges; Section D gives the full schema and Figure 3 illustrates it.

[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 1: Failures that can pass the Lean 4 type check 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` and a witness that carries no real obstruction. | A witness must carry a real obstruction and cannot be `True` or an empty structure; satisfiability is screened earlier, by the mathematical review of the proposal. |
| Unproved step | Step asserted as `axiom`; `sorry`/`admit`; a node the proof depends on mis-tagged as cited. | Automatic scan for these keywords; source check for cited nodes. |
| Over-narrow statement | Dropped hypothesis; hardcoded constant where general was intended. | Comparison with the intended claim; check for unused hypotheses. |

### 5.1 Discovery, formalization, and proof

A researcher may supply a topic anchor, or Discovery may invoke the topic-selection skill: it reads recent work, previous runs, and theoretical papers with their citing literature, ranks candidate topics by mathematical potential and by whether the result would have a concrete application, and submits the best one to an independent adversarial topic review, which must pass before the pipeline starts (Section E.1).

Once a topic 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 review compares it with the literature and judges whether it is novel, and a solver then solves the proposed open questions in natural language; their outputs become the initial graph nodes and edges. When a proposed claim is too strong or too weak, the solver may adjust it to obtain a meaningful result. A mathematically sound but insufficiently novel proposal is downgraded, whereas an incorrect or trivial one is rejected.

Formalization turns this graph into a proof plan. Each node receives an intended Lean 4 declaration, a module location, and either a statement to prove or an existing CAUSALEAN result to reuse, and the plan becomes a skeleton file in which each statement to prove is left `sorry`. A review-and-fill loop then closes them: on each iteration the reviewer repairs declarations that have drifted from the graph and fills each remaining `sorry`, using the search engine of Section 4.2 to reuse existing results. An automatic check flags hypotheses the proof never uses, which may indicate a vacuous or mis-stated theorem.

### 5.2 Statement matching

The statement-match review in Figure 1 is where the pipeline guarantees faithfulness. As Section 3 notes, the Lean 4 type check says nothing about whether the formal statement is the one the researcher meant to prove: a formalization can carry no proof gaps 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. This review checks each formal statement against the intended claim stored in its graph node, the criterion being logical equivalence, with neither side stronger nor weaker than the other. Table 1 groups the failure modes into four families and shows where the pipeline catches each one.

The review runs in two layers. The first is mechanical: a scan rejects any completed artifact containing `axiom`, `sorry`, `admit`, `opaque`, or `unsafe`, and applies the same test to every library module the artifact depends on, directly or indirectly. In the second, several agents read the Lean 4 declaration text extracted from the compiled file, compare it against the intended statement, and flag any divergence as drift, requiring every claimed witness to carry a concrete obstruction and every cited node to have a real source match. Once every statement has a proof, two independent models recheck the full frozen graph, including nodes the latest changes did not touch, without reference to the earlier review. This is the final review in Figure 1 before the result enters the presentation stage.

### 5.3 LIBRARY FEEDBACK AND PRESENTATION

Two feedback channels connect one run to future runs: a run history of what has been attempted, and a verified library of what has been proved.

The run history preserves every run regardless of its outcome, together with its proposal, mathematical core, and review verdicts (Table 2 and section J). Before a new proposal is drafted, Discovery searches it for open gaps and near-duplicates, and the novelty-and-duplication review 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.

The library channel handles reusable lemmas during the run. When the graph identifies a missing reusable lemma, the pipeline can prove it and promote it to CAUSALEAN. A library builder (called `study` mode in our pipeline) plans the lemmas and writes a file with the statements left open; a prover fills in the proofs against the live compiler; and a reviewer, backed by a build check, assesses whether the resulting lemma is generic, reusable, non-vacuous, and `sorry`-free (Section H).

The presentation stage then converts an accepted result into a working paper in which each theorem, definition, and assumption links to its verified Lean 4 source, as is established practice (Zhu et al., 2026). Each link carries both facts attached to its graph node: the declaration compiles, and its statement has been checked as matching the English claim. An LLM reviewer reads the assembled paper the way a journal referee would, treating the verified theorems as given, and returns an advisory score for the run history. The same stage publishes to the companion site, where the paper carries a proof map, links every statement to the declaration it was audited against, and ships as a slide deck, and where a signed-in reader can verify, rate, and comment on individual statements (Sections G and I).

## 6 RESULTS

We analyze the pipeline's run history and stage logs. All results reported here come from one base-model lineup, whose members were updated during the campaign: OpenAI reasoning models carry the main mathematics and the Lean 4 formalization, and Anthropic's Claude handles Lean 4 code review and planning throughout. No controlled model comparison or fixed-model ablation is reported. Section J maps every role to its model, gives the transition dates, and states what the run logs attest.

The run history holds 407 runs, distributed across three outcomes (Table 2). A run is accepted only when it is sound, novel at its requested publishable tier, and proved to completion in Lean 4; twenty-two runs meet this bar. A run that is sound but falls below its novelty target is downgraded (253 runs), and one rejected at the proposal or at the mathematics is failed (132 runs). Neither is discarded: as Section 5.3 describes, downgraded and failed runs feed the history that screens later proposals.

### 6.1 OUTCOMES BY CLUSTER AND QUESTION TYPE

Discovery files every run under one of six clusters before any mathematics begins. Sorted that way (Table 2, top), the accepted results gather in `Stat`, `Experimentation`, and `Panel`, which account for nineteen of the twenty-two acceptances, while the three identification and structural-model clusters together hold 214 runs and only three.

We also labeled each run by what its proposal is about, as a method comparison, an assumption relaxation, a gap filling, or a new setting; Section J states the rule and Table 2 (bottom) gives that cut of the catalog. All twenty-two acceptances are gap filling. The remaining three types each fail in a characteristic way. Assumption-relaxation runs usually reach a correct derivation and then lose on novelty, giving 42 downgrades against 24 failures; reviewers describe what comes back as "generic Manski-style outer containment" or "while correct, a standard finite-dimensional support/projection calculation," so the difficulty lies in conception rather than in the proof. Method-comparison and new-setting runs stop earlier, 13 of 19 and 32 of 50 failing outright, often at the proposal stage.

Table 2: The 407-run catalog, cut two ways. *Top*: by the cluster Discovery assigned at proposal time. *Bottom*: by question type, assigned by the rule of Section J from the proposal-time topic string each run carries; we assigned these labels once the runs had finished. "Accept rate" is accepted runs over total runs in the row.

| | Acc. | Down. | Fail. | Total | Accept rate |
|---|---|---|---|---|---|
| *By discovery cluster* | | | | | |
| `Stat` | 12 | 59 | 24 | 95 | 13% |
| `Experimentation` | 6 | 47 | 10 | 63 | 10% |
| `Panel` | 1 | 22 | 12 | 35 | 3% |
| `ExactID` | 2 | 35 | 38 | 75 | 3% |
| `PartialID` | 1 | 56 | 37 | 94 | 1% |
| `SCM` | 0 | 34 | 11 | 45 | 0% |
| *By question type* | | | | | |
| Gap filling | 22 | 187 | 63 | 272 | 8% |
| Assumption relaxation | 0 | 42 | 24 | 66 | 0% |
| New setting | 0 | 18 | 32 | 50 | 0% |
| Method comparison | 0 | 6 | 13 | 19 | 0% |
| Total | 22 | 253 | 132 | 407 | 5% |

Gap-filling proposals often have a more quantitative target, such as a better convergence rate or a matched lower bound, while the other three types do not.

**Where the gaps come from.** Some gaps are explicitly stated in the literature, and some the system finds for itself. We read the anchor paper behind each of the twenty-two accepted results. Three inherit a gap the anchor states in its own text, as when Zeng et al. (2024) leave the closure of their $\log^2$ gap to future work or Cortez-Rodriguez et al. (2023) trace the degree gap between their bounds to "a weakness in the analyses." The other nineteen aim at a quantity the literature had yet to ask about, such as the pseudo-true estimand of staggered PPML (Moreau-Kastler, 2025) or the minimax confidence-set length for a transported complier average causal effect (Chen & Huang, 2025).

### 6.2 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. (2024) established a minimax lower bound of order $n^{-1} + \big(d/(n \log n)\big)^2$ 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 $\epsilon \le \Pr(A = 1 \mid X = k) \le 1 - \epsilon$, but the estimators they analyze leave a $\log^2$-factor gap between the known upper and lower bounds. The run constructs an 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 and uses a pilot count to label each covariate cell heavy or light, giving heavy, well-populated cells a plug-in ratio estimator and light, sparsely sampled cells a best-polynomial approximation of the cell functional with unbiased factorial-moment lifting. The headline theorem states that, for every fixed overlap $0 < \epsilon < 1/2$, this estimator attains

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

so the minimax MSE has parametric order $n^{-1}$ whenever $d = O(\sqrt{n} \log n)$ and tends to zero if and only if $d = o(n \log n)$.

The novel part is the hybrid estimator and its matched rate; the minimax lower half is not checked here but cited from the moment-matching bound of Zeng et al. (2024). In the Lean 4 code that imported bound appears as an explicit hypothesis on the headline theorem, following the cited-result discipline of Table 1: an external result the proof depends on is carried as a hypothesis with source evidence, so a reader sees which part is proved in this work and which is from the literature.

### 6.3 MACHINE-CHECKED SOUNDNESS

Each accepted result is screened for the unproved-step failures of Table 1 before it enters the run history. Starting from a clean build, we ask Lean 4 which axioms the headline theorem depends on, and cross-check by scanning the run's module and every library module it reaches for `sorry`, `admit`, and `axiom` (Section J). For every accepted paper the formalization contains none of these and the theorems reduce to Lean 4's standard axioms; the flagship result's single external input, the Zeng et al. (2024) lower bound, is visible as a hypothesis.

### 6.4 LIBRARY FEEDBACK IN ACTION

The flagship result of Section 6.2 gives a complete instance of the library feedback loop. Its estimator labels each covariate cell heavy or light by a pilot count, so the proof needs two-sided multiplicative Chernoff tails for the count of an i.i.d. $[0, 1]$-valued statistic, which Mathlib does not have. The run proved them, and promotion moved them into CAUSALEAN as `Causalean.Stat.Concentration.TailBounds.BinomialCount`, whose `bernoulliCount_upper_tail` and `bernoulliCount_lower_tail` the run then applies. The next accepted result on the same estimand consumed that promotion: a two-sided minimax bracket under a shrinking heterogeneity radius needs the pilot argument over an arbitrary finite category set where the flagship had two cell types, so the run built `Causalean.Stat.SampleSplit.FiniteCategoryPilot`, which imports `BinomialCount` and reuses its moment-generating-function bound. A run required a lemma, the system proved it, promotion added it to the library, and a later run on the same problem built on it instead of repeating it. Eighty-four completed library builds have produced promotions of their own, including Fano's inequality, a Bretagnolle–Huber affinity bound for arbitrarily many hypotheses, a KL density-tilt expansion, and the Ehlich–Zeller and Bernstein–Szegő tools behind an accepted Chebyshev rollout design. Estimating how library growth trades off against cost per result remains future work (Section 7).

## 7 DISCUSSION AND LIMITATIONS

Automated theoretical research needs a reliable basis for judging correctness, and LLM reviewers can be misled. CAUSALSMITH answers that at three separate levels of confidence: the Lean 4 type check settles whether a proof is valid, the statement audit guarantees that the Lean 4 statement matches the intended claim, and whether a finished paper matters stays a human judgment. Keeping the three apart is what stops a valid proof from being mistaken for an important result. Only the formal library is specific to causal inference; moving to another field means building or adopting a comparable library and retrieval interface, while the graph, the audit, and the library feedback loop should carry over with little change.

Our evaluation has two main limitations: research quality is rated by the pipeline itself, with no independent human validation of those LLM ratings, and the statement-match audit is itself an LLM task, which frontier models handled well in our observations but whose error rate we do not estimate. Taken together, the verified library, the self-directed pipeline, and the feedback loop that grows it give machine-checked proofs of self-proposed results with audited statements. We leave better novelty-evaluation metrics and faithfulness-audit benchmarks to future work.

## AI USE STATEMENT

Generative AI is part of both the system studied and its development. As described in Sections 4 and 5, language models assist with formal-library development, propose research questions and mathematical claims, draft informal derivations and formal proofs, review statement alignment, and generate research presentations. The model roles and campaign changes are documented in Appendix J. Language-model assistance was also used to adapt this manuscript to the conference format, prepare the supplementary package and polish the final version. Lean checks formal proofs; this check does not establish novelty, faithfulness to an informal claim, or the validity of empirical interpretation. The authors are responsible for the submission's text, claims, code, and artifacts.

## REPRODUCIBILITY STATEMENT

The anonymous supplementary package contains the pinned Lean projects, agent implementation and prompts, the labeled 407-run campaign manifest, and the redacted run history. Its evaluation script reconstructs the campaign outcome, cluster, and question-type tables and reports differences between campaign labels and current bank dispositions. Build instructions, verification scope, and requirements for live model calls are documented in the package README. Appendix J describes the campaign and audits.

## REFERENCES


Sawal Acharya, Rahul Shrestha, Jacob Emmerson, Vishal Verma, Devansh Bhardwaj, Samuel Simko, Punya Pandey, Anahita Haghighat, Yongjin Yang, Dominik Janzing, Mrinmaya Sachan, Bernhard Schölkopf, and Zhijing Jin. Causal AI scientist: Towards end-to-end causal inference with large language models. In *ICML 2026 Workshop on AI as a Tool for Mathematics, Computer Science, and Machine Learning*, 2026a. URL https://openreview.net/forum?id=rIAuqhy6S0.

Sawal Acharya, Terry Jingchen Zhang, Andrew Kim, Rahul Babu Shrestha, Xianlin Sun, Pepijn Cobben, Maximilian Mordig, Jacob T. Emmerson, Anahita Haghighat, Furkan Danisman, Yuen Chen, Clijo Jose, Andrei Ioan Muresanu, Justin Cui, Jiarui Liu, Yahang Qi, Punya Syon Pandey, Yinya Huang, Bernhard Schölkopf, and Zhijing Jin. CauSciBench: Can LLMs automate causal inference in real-world scientific research? In *International Conference on Machine Learning (ICML)*, 2026b. URL https://openreview.net/forum?id=v9d4RqaPFO.

Daneshvar Amrollahi, Jerry Lopez, and Clark Barrett. Faithful autoformalization via roundtrip verification and repair. *arXiv preprint arXiv:2604.25031*, 2026. doi: 10.48550/arXiv.2604.25031. URL https://arxiv.org/abs/2604.25031v2.

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. doi: 10.48550/arXiv.2302.12433. URL https://arxiv.org/abs/2302.12433.

Joeran Beel, Min-Yen Kan, and Moritz Baumgart. Evaluating Sakana's AI scientist: Bold claims, mixed results, and a promising future? *ACM SIGIR Forum*, 59(1):1–20, 2025. doi: 10.1145/3769733.3769747. URL https://doi.org/10.1145/3769733.3769747. arXiv:2502.14297.

Rafael Cabral, Tuan Manh Do, Xuejun Yu, Wai Ming Tai, Zijin Feng, and Xin Shen. ProofFlow: A dependency graph approach to faithful proof autoformalization. *arXiv preprint arXiv:2510.15981*, 2025. doi: 10.48550/arXiv.2510.15981. URL https://arxiv.org/abs/2510.15981v1.

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. doi: 10.48550/arXiv.2405.00622. URL https://arxiv.org/abs/2405.00622.

Zhongren Chen and Melody Huang. Generalizing causal effects with noncompliance: Application to deep canvassing experiments. *arXiv preprint arXiv:2506.00149*, 2025. doi: 10.48550/arXiv.2506.00149. URL https://arxiv.org/abs/2506.00149v2. Version 2, revised June 6, 2026.

Joanie Hayoun Chung, Sumin Lee, and Sungbin Lim. ORCA: ORchestrating causal agent. In *Extended Abstracts of the 2026 CHI Conference on Human Factors in Computing Systems*. ACM, 2026. doi: 10.1145/3772363.3798444. URL https://doi.org/10.1145/3772363.3798444.

Mayleen Cortez-Rodriguez, Matthew Eichhorn, and Christina Lee Yu. Exploiting neighborhood interference with low order interactions under unit randomized design. *Journal of Causal Inference*, 11(1):20220051, 2023. doi: 10.1515/jci-2022-0051. URL https://doi.org/10.1515/jci-2022-0051. arXiv:2208.05553.

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. doi: 10.48550/arXiv.2606.28013. URL https://arxiv.org/abs/2606.28013v1.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction (CADE 28)*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37. URL https://link.springer.com/chapter/10.1007/978-3-030-79876-5_37.

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. doi: 10.48550/arXiv.2303.04910. URL https://arxiv.org/abs/2303.04910.

Jinglong Gao, Xiao Ding, Bing Qin, and Ting Liu. Is ChatGPT a good causal reasoner? a comprehensive evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 11111–11126. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.743. URL https://aclanthology.org/2023.findings-emnlp.743/. arXiv:2305.07375.

Nikhil Garg. EconCSLib: AI-assisted lean formalization for economics & computation research. *arXiv preprint arXiv:2606.13306*, 2026. doi: 10.48550/arXiv.2606.13306. URL https://arxiv.org/abs/2606.13306v2.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomašev, Dina Zverinski, Ivor Rendulic, Elahe Vedadi, Florian Hasler, Luka Rimanic, Marina Boia, Ivan Budiselic, Ben Feinstein, Mathias Bellaiche, Tom Sheffer, Jan Freyberg, Jeremy Ratcliff, Ottavia Bertolli, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R. D. Costa, José R. Penadés, Gary Peltz, Yossi Matias, James Manyika, Demis Hassabis, Yunhan Xu, Pushmeet Kohli, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Accelerating scientific discovery with Co-Scientist. *Nature*, 655:487–496, 2026. doi: 10.1038/s41586-026-10644-y. URL https://www.nature.com/articles/s41586-026-10644-y. Published online 19 May 2026; arXiv:2502.18864, whose v1 appeared as "Towards an AI co-scientist".

Sukjin Han. Mining causality: AI-assisted search for instrumental variables. *arXiv preprint arXiv:2409.14202*, 2024. doi: 10.48550/arXiv.2409.14202. URL https://arxiv.org/abs/2409.14202.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, Paul Lezeau, Salvatore Mercuri, Calle Sönne, Bhavik Mehta, Alex Davies, Daniel Zheng, Fabian Pedregosa, Yin Li, Ingrid von Glehn, Mark Rowland, Samuel Albanie, Ameya Velingker, Simon Schmitt, Edward Lockhart, Edward Hughes, Henryk Michalewski, Nicolas Sonnerat, Demis Hassabis, Pushmeet Kohli, and David Silver. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613, 2026. doi: 10.1038/s41586-025-09833-y. URL https://www.nature.com/articles/s41586-025-09833-y. First published online 12 November 2025; version of record 13 March 2026.

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. doi: 10.48550/arXiv.2606.13925. URL https://arxiv.org/abs/2606.13925v1.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://www.repository.cam.ac.uk/bitstreams/1fba3954-37d1-46e3-9193-0642dff92b2c/download. Accepted version, ICLR 2023; arXiv:2210.12283.

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. doi: 10.48550/arXiv.2510.18003. URL https://arxiv.org/abs/2510.18003v2. Version 2, revised June 16, 2026.

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)*, volume 36, 2023. doi: 10.52202/075280-1353. URL https://papers.nips.cc/paper_files/paper/2023/hash/631bb9434d718ea309af82566347d607-Abstract-Conference.html.

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. URL https://openreview.net/pdf?id=vqIH0ObdqL.

Nitish Joshi, Abulhair Saparov, Yixin Wang, and He He. LLMs are prone to fallacies in causal inference. *arXiv preprint arXiv:2406.12158*, 2024. doi: 10.48550/arXiv.2406.12158. URL https://arxiv.org/abs/2406.12158.

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*, 2024. URL https://openreview.net/forum?id=mqoxLkX210. arXiv:2305.00050.

Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. HyperTree proof search for neural theorem proving. *arXiv preprint arXiv:2205.11491*, 2022. doi: 10.48550/arXiv.2205.11491. URL https://arxiv.org/abs/2205.11491.

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. doi: 10.48550/arXiv.2503.18821. URL https://arxiv.org/abs/2503.18821. https://github.com/optsuite/optlib.

Zenan Li, Yifan Wu, Zhaoyu Li, Xinming Wei, Fan Yang, Xian Zhang, and Xiaoxing Ma. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1697. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/6034a661584af6c28fd97a6f23e56c0a-Abstract-Conference.html. arXiv:2410.20936.

Yongxi Lin, Debarghya Mukherjee, Rajarshi Mukherjee, Fred Rajasekaran, and Zixiao Jolene Wang. Statlib. Software library, 2026. URL https://stat-lib.github.io/. Initial contributors listed on the project website; accessed 15 September 2026.

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*, pp. 9215–9235. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.548. URL https://aclanthology.org/2024.findings-acl.548/. arXiv:2402.17644.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. doi: 10.48550/arXiv.2408.06292. URL https://arxiv.org/abs/2408.06292.

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. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/fceedf8c9c0ff51f41b9fe0294ef0070-Abstract-Conference.html. arXiv:2410.10135.

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

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. doi: 10.48550/arXiv.2606.14000. URL https://arxiv.org/abs/2606.14000v1.

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. doi: 10.48550/arXiv.2606.16541. URL https://arxiv.org/abs/2606.16541v1.

Ninon Moreau-Kastler. Proportional treatment effects in staggered settings: An approach for poisson pseudo-maximum likelihood. Working Paper 31, EU Tax Observatory, 2025. URL https://www.taxobservatory.eu/www-site/uploads/2025/05/WP31_-Proportional-Treatment-Effects-in-Staggered-Settings.pdf.

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. doi: 10.48550/arXiv.2506.13131. URL https://arxiv.org/abs/2506.13131.

Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. Reliable evaluation and benchmarks for statement autoformalization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17947–17969. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.907. URL https://aclanthology.org/2025.emnlp-main.907/. arXiv:2406.07222.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020. doi: 10.48550/arXiv.2009.03393. URL https://arxiv.org/abs/2009.03393.

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. doi: 10.48550/arXiv.2504.21801. URL https://arxiv.org/abs/2504.21801.

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. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

Ayush Sawarni, Jiyuan Tan, and Vasilis Syrgkanis. Causalreasoningbenchmark: A real-world benchmark for disentangled evaluation of causal identification and estimation. arXiv preprint arXiv:2602.20571, 2026. URL https://arxiv.org/abs/2602.20571.

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. doi: 10.48550/arXiv.2412.13667. URL https://arxiv.org/abs/2412.13667.

Ivaxi Sheth, Zhijing Jin, Bryan Wilder, Dominik Janzing, and Mario Fritz. IV co-scientist: Multi-agent LLM framework for causal instrumental variable discovery. In *Proceedings of the Fifth Conference on Causal Learning and Reasoning*, volume 323 of *Proceedings of Machine Learning Research*, pp. 1293–1317. PMLR, 2026. URL https://proceedings.mlr.press/v323/sheth26a.html.

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. doi: 10.48550/arXiv.2404.12534. URL https://arxiv.org/abs/2404.12534.

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. doi: 10.48550/arXiv.2503.19605. URL https://arxiv.org/abs/2503.19605. https://github.com/auto-res/lean-rademacher.

StatsMLlib contributors. StatsMLlib: Verified probability, statistics, and learning theory in Lean 4. Software library, 2026. URL https://github.com/Lean-MoDS/StatsMLlib. Project-level attribution; organizers and contributors distinguished in AUTHORS.md; accessed 15 September 2026.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, pp. 367–381. ACM, 2020. doi: 10.1145/3372885.3373824. URL https://leanprover-community.github.io/papers/mathlib-paper.pdf.

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. URL https://www.nature.com/articles/s41586-023-06747-5.

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)*, 37, 2024. doi: 10.52202/079017-0368. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/1582eaf9e0cf349e1e5a6ee453100aa1-Abstract-Datasets_and_Benchmarks_Track.html. arXiv:2407.11214.

George Tsoukalas, Anton Kovsharov, Sergey Shirobokov, Anja Surina, Moritz Firsching, Gergely Bérczi, Francisco J. R. Ruiz, Arun Suggala, Adam Zsolt Wagner, Eric Wieser, Lei Yu, Aja Huang, Miklós Z. Horváth, Andrew Ferraiuolo, Henryk Michalewski, Edward Lockhart, Codrut Grosu, Thomas Hubert, Matej Balog, Pushmeet Kohli, and Swarat Chaudhuri. Advancing mathematics research with AI-driven formal proof search. *arXiv preprint arXiv:2605.22763*, 2026. doi: 10.48550/arXiv.2605.22763. URL https://arxiv.org/abs/2605.22763.

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. doi: 10.48550/arXiv.2605.10141. URL https://arxiv.org/abs/2605.10141v1.

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. doi: 10.48550/arXiv.2504.13263. URL https://arxiv.org/abs/2504.13263.

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. doi: 10.48550/arXiv.2606.20642. URL https://arxiv.org/abs/2606.20642v1.

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)*, volume 35, 2022. doi: 10.52202/068431-2344. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/d0c6bc641a56bebee9d985b937307367-Abstract-Conference.html.

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. doi: 10.48550/arXiv.2504.08066. URL https://arxiv.org/abs/2504.08066.

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*, volume 36, 2023. doi: 10.52202/075280-0944. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html.

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*, 2024a. doi: 10.48550/arXiv.2412.16075. URL https://arxiv.org/abs/2412.16075v1.

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*, 2024b. doi: 10.48550/arXiv.2407.08029. URL https://arxiv.org/abs/2407.08029.

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. URL https://openreview.net/forum?id=tv46tCzs83. arXiv:2308.13067.

Zhenghao Zeng, Sivaraman Balakrishnan, Yanjun Han, and Edward H. Kennedy. Causal inference with high-dimensional discrete covariates. *arXiv preprint arXiv:2405.00118*, 2024. doi: 10.48550/arXiv.2405.00118. URL https://arxiv.org/abs/2405.00118v3. Version 3, revised June 18, 2026.

Yuanhe Zhang, Jason D. Lee, and Fanghui Liu. AI4SLT: Empirical processes in Lean 4 for formal statistical learning theory. *arXiv preprint arXiv:2602.02285*, 2026. doi: 10.48550/arXiv.2602.02285. URL https://arxiv.org/abs/2602.02285.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. MiniF2F: A cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=9ZPegFuFTFv. arXiv:2109.00110.

Thomas Zhu, Pietro Monticone, Jeremy Avigad, and Sean Welleck. LeanArchitect: Automating blueprint generation for humans and AI. *arXiv preprint arXiv:2601.22554*, 2026. doi: 10.48550/arXiv.2601.22554. URL https://arxiv.org/abs/2601.22554v1.

# A EXTENDED RELATED WORK

This appendix gives the four lines of work of Section 2 at full length.

Table 3: Where CAUSALSMITH 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 (Yamada et al., 2025) | ✓ | — | — | LLM judge | — |
| FunSearch / AlphaEvolve (Romera-Paredes et al., 2024; Novikov et al., 2025) | partial | — | — | task metric | ✓ |
| AlphaProof (Hubert et al., 2026) | — | ✓ | n/a | sound | partial |
| Causal benchmarks (Jin et al., 2023; 2024) | — | — | — | fixed labels | — |
| Causal agents (Wang et al., 2025) | — | — | — | none | — |
| Lean economics library (Garg, 2026) | — | ✓ | ✓ | sound + audit | — |
| Asymptotic-statistics agents (Wei et al., 2026) | — | ✓ | ✓ | sound + audit | — |
| **CAUSALSMITH (ours)** | ✓ | ✓ | ✓ | sound + audit | ✓ |

**Autoformalization, theorem proving, and statement matching.** Language models have been paired with proof assistants for neural proof search (Polu & Sutskever, 2020; Lample et al., 2022), autoformalization (Wu et al., 2022; Jiang et al., 2023), whole-proof generation and repair (First et al., 2023), retrieval-augmented proving (Yang et al., 2023), reinforcement learning with subgoal decomposition (Ren et al., 2025), and competition mathematics (Trinh et al., 2024; Hubert et al., 2026), and they are measured against benchmarks of known problems (Zheng et al., 2022; Azerbayev et al., 2023; Tsoukalas et al., 2024). Closest to us is autoformalization, where a model turns an informal statement into a formal one. Evaluation methods include expert inspection (Wu et al., 2022), back-translation and semantic consistency checks (Li et al., 2024), probe-based faithfulness checks (Mohammad & Sheikh, 2026), and learned alignment scorers (Lu et al., 2025). Roundtrip verification has also been studied for statutory text without ground-truth formal annotations (Amrollahi et al., 2026). Surface overlap alone is insufficient to establish semantic correctness (Poiroux et al., 2025). In our setting, the theorem is new, so the run writes its own informal claim and then formalizes it, leaving both sides of the comparison model output. Studies of this gap report formalizations with no remaining proof gaps that still fail expert review (Ilin & Nugent, 2026), and statements a proof assistant accepts that are semantically wrong (Dai et al., 2026; Meek et al., 2026). A separate benchmark tests whether learned proof evaluators distinguish correct proofs from injected incorrect variants (Uluşan et al., 2026). Our audit therefore back-translates, inside the discovery loop and localized over a proof-dependency graph, in the style of Lean blueprints and proof-flow tools (Zhu et al., 2026; Cabral et al., 2025), within the broader shift toward proof-assistant-grounded reasoning (Yang et al., 2024a).

**Automated research agents and verified discovery.** Agents that carry out end-to-end research (Yamada et al., 2025) are vulnerable when an LLM judges their output, whether shown by independent re-evaluation (Beel et al., 2025) or by adversarial construction (Jiang et al., 2025). Other work grounds discovery in a hard verifier: program search against a fitness function (Romera-Paredes et al., 2024; Novikov et al., 2025), or a proof against a type checker. Those fitness functions are fixed in advance, so a statement audit is a step they never need. "Co-scientist" agents fall in between: they propose and refine scientific claims but validate them empirically (Gottweis et al., 2026). The system closest to ours is a multi-agent pipeline for asymptotic statistics, whose auditor and reviewer agents check for unsupported strengthening of hypotheses (Wei et al., 2026). It formalizes known theory, whereas CAUSALSMITH proposes the theorem as well, and CAUSALSMITH localizes the audit over the dependency graph, so a changed statement re-opens the affected nodes rather than the whole formalization.

**LLMs for causal inference.** A parallel literature asks whether language models reason causally, treating causal reasoning as something to test or deploy rather than something to prove (Kıcıman et al., 2024); recent surveys organize it by causal task and intervention level (Ma, 2025). Benchmarks probe graphical and correlational reasoning (Jin et al., 2023; 2024), broad and data-grounded question answering (Chen et al., 2024; Liu et al., 2024), and, closest to our own domain, end-to-end causal inference on real scientific studies (Acharya et al., 2026b; Sawarni et al., 2026). A critical strand reports that surface accuracy conceals shallow reasoning: models explain causal language while hallucinating the argument (Gao et al., 2023), and commit elementary fallacies in controlled tests (Joshi et al., 2024). Other work questions whether causal language reflects an underlying causal model (Zečević et al., 2023) and warns that some benchmarks may be solved through knowledge retrieval (Yang et al., 2024b). A third strand builds agents that do causal analysis without producing machine-checked causal theorems (Shen et al., 2024; Chung et al., 2026; Wang et al., 2025), including pipelines that map a dataset and question to an estimate (Acharya et al., 2026a) and systems that search for instrumental variables (Han, 2024; Sheth et al., 2026). The IV Co-Scientist uses a propose-and-critique structure (Sheth et al., 2026); these causal systems evaluate outputs statistically or empirically. Lean Copilot pairs language models with formal verification for general mathematics (Song et al., 2024), and CAUSALSMITH brings this approach to causal-inference research.

**Formal libraries for statistics and economics.** A Lean 4 ecosystem for theoretical statistics has recently emerged: Statlib on the decision-theoretic foundations of inference (Lin et al., 2026), StatsMLlib on concentration, empirical processes, and finite-sample learning guarantees (StatsMLlib contributors, 2026), and a statistical-learning-theory line (Zhang et al., 2026) extending the Rademacher-complexity library that CAUSALEAN adapts (Sonoda et al., 2025). A Lean 4 library for economics formalizes established theory in the same spirit (Garg, 2026). CAUSALEAN overlaps with all of them wherever causal estimation theory needs the same tools—minimax lower-bound methods, information divergences, empirical-process and concentration lemmas—and we expect these projects to benefit as they converge on shared upstream infrastructure. Those libraries formalize classical theory as the end product, whereas CAUSALEAN's statistical clusters exist to support theorems about causal estimands, and its core—structural causal models, do-calculus, graphical and partial identification, panel estimand characterizations, and design-based inference under interference—has no counterpart in any of them. CAUSALEAN is also built to be extended rather than finished: each accepted run can promote its reusable lemmas back into the library for later runs to use.

## B FLAGSHIP THEOREMS

Table 4 catalogs the main result families across CAUSALEAN's major research areas (Section 4.1). Each entry is a named declaration with a machine-checked proof, listed with its source location so that a reader can look it up directly.

Table 4: Flagship theorems across CAUSALEAN's major research areas. Each entry is a named, machine-checked declaration; the third column gives its source location, relative to the `Causalean/` package root at the pinned toolchain `leanprover/lean4:v4.29.0-rc3`. Long identifiers and paths may wrap.

| Result | Declaration | Source |
|---|---|---|
| *Graphical and structural causal models* | | |
| Markov equivalence iff two DAGs have the same skeleton and immoralities | `markovEquiv_iff_sameSkeleton_sameImmoralities` | `Graph/MarkovEquiv.lean:51` |
| Rule 2 of do-calculus under its graphical condition | `do_rule2_kernel` | `SCM/Do/DoCalculus.lean:95` |
| Backdoor adjustment identifies an interventional distribution almost everywhere | `backdoor_identifiable_ae` | `SCM/ID/Backdoor.lean:424` |
| Frontdoor adjustment identifies an interventional distribution almost everywhere | `frontdoor_identifiable_ae` | `SCM/ID/Frontdoor.lean:1169` |

*Table 4 continued*

| Result | Declaration | Source |
|---|---|---|
| Soundness of the graphical ID algorithm on the discrete positive class | `id_sound_discrete` | `SCM/ID/GraphicalThms/IDSoundDiscrete.lean:30` |
| *Causal discovery* | | |
| LiNGAM identifiability from nonzero source kurtosis | `lingam_identifiability_kurtosis` | `Discovery/LiNGAM/LiNGAMKurtosis.lean:73` |
| Soundness of invariant causal prediction | `icp_sound` | `Discovery/InvariantPrediction/Soundness.lean:34` |
| Completeness of invariant prediction for linear-Gaussian models | `icp_complete_linearGaussian` | `Discovery/InvariantPrediction/.../Completeness.lean:288` |
| Identifiability in linear causal disentanglement | `disentanglement_identifiability` | `Discovery/LinearDisentanglement/Identifiability.lean:35` |
| *Potential outcomes, exact identification, and partial identification* | | |
| Wald-ratio identification of the local average treatment effect | `late_wald` | `PO/ID/Exact/LATE.lean:452` |
| Difference-in-differences identification of the ATT | `att_did` | `PO/ID/Exact/DID.lean:173` |
| Callaway–Sant'Anna group-time ATT identification | `att_csdid` | `PO/ID/Exact/CSDID.lean:463` |
| Sharp regression-discontinuity identification | `rdd_identification` | `PO/ID/Exact/RDD/SharpRDD.lean:316` |
| Fuzzy regression-discontinuity identification | `frd_identification` | `PO/ID/Exact/RDD/FuzzyRDD.lean:606` |
| Wald identification for dynamic treatment timing | `whenToTreat_wald` | `PO/ID/Exact/DynamicLATE/WhenToTreat.lean:331` |
| Manski worst-case ATE bounds | `manski_bounds_ATE` | `PO/ID/Partial/Manski/NonAsp.lean:72` |
| Sharpness of the Balke–Pearl bounds for a binary instrument | `balkePearl_sharp` | `PO/ID/Partial/BalkePearl/Sharp.lean:856` |
| Lee bounds for the ATT under selection | `lee_bounds_ATT_AS` | `PO/ID/Partial/Lee/Main.lean:33` |
| Pointwise Imbens–Manski coverage for partially identified parameters | `imbensManski_pointwise_coverage` | `PO/ID/Partial/Inference/ImbensManski.lean:171` |
| *Estimation and statistical theory* | | |
| Asymptotic normality of the debiased-ML ATE estimator | `dml_ATE_tendstoNormal` | `Estimation/ATE/DML.lean:936` |
| Attainment of the Hahn efficiency bound by debiased-ML ATE | `dml_ATE_attains_hahn_bound` | `Estimation/Efficiency/ATEVariance.lean:426` |
| Asymptotic linearity of debiased-ML ATT estimation | `dml_ATT_isAsymLinear` | `Estimation/ATT/DML.lean:239` |
| Asymptotic normality of partially linear DML | `plr_dml_tendstoNormal` | `Estimation/PLR/DML.lean:171` |
| Asymptotic linearity of sequential doubly robust DTR estimation | `seqDR_dml_isAsymLinear` | `Estimation/DTR/DTRInstance.lean:155` |
| Structure-agnostic minimax lower bound for ATE estimation | `minimax_lower_bound_var_causal` | `Estimation/MinimaxATE/Causal/Minimax.lean:170` |
| Optimal weighting for generalized method of moments | `gmm_efficiency` | `Stat/GMM/VarianceAlgebra.lean:105` |
| Central limit theorem for regular order-$m$ U-statistics | `uStatisticOrder_clt` | `Stat/UStatistic/OrderM/CLT.lean:52` |
| *Panel and event-study methods* | | |

*Table 4 continued*

| Result | Declaration | Source |
|---|---|---|
| Causal decomposition of staggered-adoption TWFE into weighted contrasts | `twfe_po_decomposition` | `Panel/.../CausalDecomposition.lean:84` |
| Characterization of linear-unbiased BJS imputation estimators | `bjs_linear_unbiased_iff_imputation_form` | `Panel/.../ImputationEventStudy/PanelBridge.lean:360` |
| Event-study pretrends induced by post-treatment effects | `apparent_pretrends_from_post_treatment_of_cellGrid` | `Panel/.../EventStudyContamination/Contamination.lean:75` |
| *Experimentation under interference* | | |
| Consistency of Horvitz–Thompson estimation under unknown interference | `htEst_consistent_eate` | `Experimentation/UnknownInterference/Consistency.lean:147` |
| Central limit theorem for a two-stage interference direct effect | `directEffect_clt` | `Experimentation/.../Asymptotic/CLT.lean:118` |
| Stein-method CLT for estimators under network interference | `localDependenceCLT_of_stein` | `Experimentation/.../SteinInstance.lean:181` |
| Wald coverage under network interference | `wald_coverage_of_stein` | `Experimentation/.../SteinInstance.lean:264` |

## C THE CLUSTERS AND THE RETRIEVAL INDEX

This appendix expands Section 4.1: it walks CAUSALEAN's ten clusters in turn and then describes the retrieval interface the pipeline queries.

Table 5: 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, instances, and inductive types. The `Mathlib (local)` row also carries the tactic and simp-set support files.

| Cluster | Files | Lines | Defs | Thms/Lem. | s/c/i |
|---|---|---|---|---|---|
| Stat | 324 | 87,276 | 578 | 1,772 | 70 |
| Mathlib (local) | 256 | 66,274 | 665 | 1,578 | 98 |
| Estimation | 168 | 56,486 | 311 | 638 | 111 |
| PO | 138 | 44,470 | 597 | 1,058 | 79 |
| SCM | 88 | 37,562 | 211 | 555 | 66 |
| Panel | 68 | 19,318 | 273 | 423 | 47 |
| Experimentation | 105 | 18,347 | 188 | 374 | 12 |
| Graph | 37 | 15,234 | 114 | 278 | 37 |
| Discovery | 35 | 10,061 | 90 | 175 | 12 |
| ML | 50 | 6,618 | 56 | 96 | 13 |
| **Total** | **1,269** | **361,646** | 10,575 **declarations** | | |

### C.1 A TOUR OF THE CLUSTERS

**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 $\mathrm{do}$-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. These results take a causal query to a formula in the observational distribution, or to a nonidentification proof.

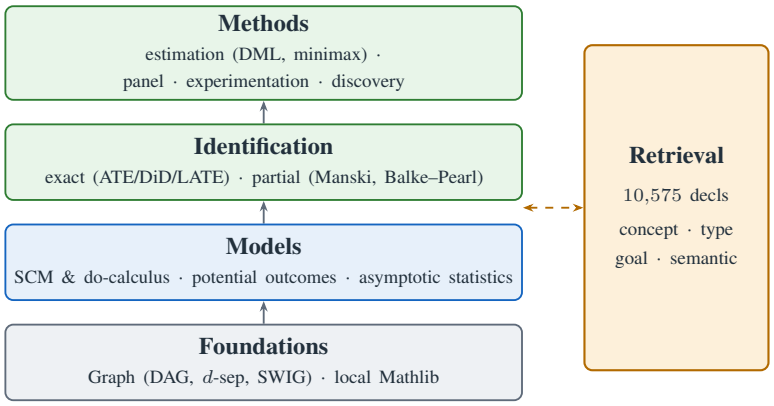



Figure 2: CAUSALEAN in layers: graphical and measure-theoretic foundations support the model languages, which support identification, which supports the estimation and design methods on top. A retrieval index over all 10,575 declarations spans the stack and is the interface the pipeline queries.

**Potential outcomes and identification.** The PO cluster holds the most definitions of the causal clusters, 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 provides tight bounds for estimands that cannot be point-identified: 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, and marginal-sensitivity models.

**Estimation and asymptotic statistics.** Estimation 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 with localization, and the bootstrap. The localized theory covers star hulls, sub-root envelopes, local Rademacher complexity, and the critical radius that controls a localized uniform-deviation bound.

**Panel, experiments, and discovery.** Panel builds general machinery for panel data: potential outcomes indexed by treatment path, fixed effects, and least squares as projection with a Frisch–Waugh–Lovell decomposition. Using that machinery, it 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 covers 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.

### C.2 RETRIEVAL

For agents we build a search engine on top of that index, and we train a retrieval encoder on CAUSALEAN itself to drive part of it. The training pairs come from the keyword index: a theorem's docstring against the declarations it is built on, and each declaration's Lean 4 statement against its own docstring. The split is by module, so the encoder is evaluated only on modules it never saw. All 10,575 declarations are then embedded into a vector index of 1,024 dimensions.

Search runs in three modes. A concept mode expands a natural-language query with causal-inference synonyms. A type-pattern mode matches a fragment of Lean 4 syntax against the symbols in each declaration's statement. A goal-directed mode ranks candidates against the goal a proof is stuck on. Concept and goal queries are ranked twice, once by keyword score and once by embedding similarity, and the two rankings are combined by reciprocal-rank fusion. A fine-tuned cross-encoder then reorders the leading concept candidates. Type-pattern queries use the keyword index alone.

## D DEPENDENCY-GRAPH SCHEMA

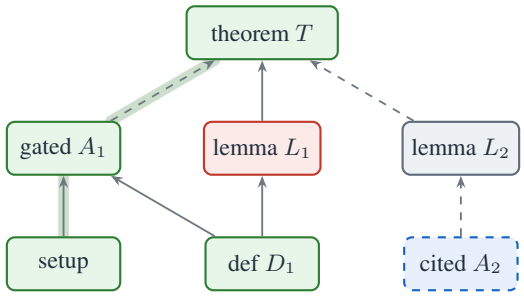



Figure 3: A dependency graph (schematic). Nodes are statements; fill encodes review status (**matched**, **drift**, **unreviewed**), and a dashed blue border marks a cited node. Solid edges show what a claim says, dashed edges what its proof uses; the green underlay is the critical path of must-prove nodes the review must clear.

Each result carries one dependency graph (Section 5), stored on disk as a `FormalizationGraph`. A node has a kind, one of `setup`, `definition`, `assumption`, `lemma`, `theorem`, or `gate`. It also holds its natural-language statement, its Lean 4 declaration, a review status (`unreviewed`, `matched`, or `drift`), and a `gate_class` field, either `gated` or `cited`, the must-prove and cited classes of Section 5. An assumption node takes one further label, saying whether the assumption refines the claim, does regularity bookkeeping, or stands in for a library gap. Edges are typed as `statement-uses`, `proof-uses`, or `setup-of`, and store their two endpoints. Before a graph is written, a validator checks it: duplicate nodes and edges are rejected, and any structural invariant that has been violated is reported.

## E IMPLEMENTATION DETAILS OF THE PIPELINE

Here we describe the pipeline of Section 5 as it was actually implemented for the runs we report. A theorem run is named by a question identifier together with a specialization, and the orchestrator walks it through the ordered stages of Figure 4 and table 6. The numbered stages are a finer version of the four stages of Section 5. Discovery is D-1.1 through D0.5. Formalization is F1, F1.5, and F2, and proof construction is F3. The statement audit of Section 5.2 runs at three points of the proof loop, namely F2.5, F3.5, and F4. F5 prepares the finished artifact, which the presentation state machine of Section G then picks up as a separate process. The subsections after Table 6 describe topic selection and the discovery reviews in more detail.

Table 6: Stages of a CAUSALSMITH theorem run. The internal state stores the numerical identifiers alone; the D and F prefixes are what the command-line interface and the logs use to tell discovery apart from formalization.

| Stage | Purpose | Action and acceptance condition | Principal durable output |
|---|---|---|---|
| D-1.1 | Literature survey | Searches open problems and earlier proposals for a question worth solving, before anything is proposed. | Gap list |

*Table 6 continued*

| Stage | Purpose | Action and acceptance condition | Principal durable output |
|---|---|---|---|
| D-1.2 | Proposal construction | Writes a typed proposal together with its mathematical core, which names the objects, the atomic assumptions, the statements, and how they depend on one another. | Proposal and typed core |
| D-0.5 | Proposal review | Reviews the proposal for novelty, for duplication, and for whether its mathematical direction is viable. If the proposal is rejected, or if a finding looks repairable, it goes back to whichever discovery work can address it, and formalization does not begin. | Review decisions and revision log |
| D0 | Mathematical derivation | Derives the proposed result and writes the research note. Where the derivation warrants it, the solver may narrow a claim that turns out to be too strong, provided what it hands on for the later Lean 4 proof is still a real scientific claim. | Derivation note and updated core |
| D0-max | Maximality checkpoint | Once the derivation goes through cleanly, halts the run and puts the whole-paper question to an external reviewer: is a sharper bound, a better construction, a stronger reframing, or a tighter constant available? A concrete improvement comes back to D0 as a directive, while a reviewer who confirms the result is already maximal releases the run into D0.5. | Maximality decision (verbatim reviewer finding logged) |
| D0.5 | Soundness and novelty review | Keeps a fresh re-derivation of the mathematics separate from the decisions about structure, novelty, and tier. A further cold review is added when the target tier calls for one. | Mathematical, structural, and tier reviews |
| ckpt D/F | D0.5→F1 go/no-go | Halts the run once D0.5 has passed. The orchestrator weighs whether a result that is now maximized and cleared for novelty is worth the expense of F1–F5. Resuming enters F1; stopping either files the discovery-only result or downgrades it. | Commit decision and lease re-grant |
| F1 | Formalization plan | Maps every core node, and the ambient setup it needs, onto a planned Lean 4 object. Where a result can be reused, the plan names the declaration and the module it lives in; where a needed fact is unavailable, the plan notes an openly disclosed obligation to be discharged. | `plan.json` and dependency graph |

*Table 6 continued*

| Stage | Purpose | Action and acceptance condition | Principal durable output |
|---|---|---|---|
| F1.5 | Plan and reuse review | Runs deterministic checks on coverage, node kind, dependency closure, whether each declaration it cites exists, and where modules are placed, then reviews how well the reuse fits and whether the abstraction level is right. A plan that comes back clean can stop at the first checkpoint so an operator may audit depth, reuse, and statement fidelity; in automatic mode it resumes on its own. | Plan and reuse reviews |
| F2 | Lean 4 scaffold | Writes or revises a Lean 4 scaffold. The first scaffold may still contain proof obligations. A later revision touches only the declarations that were flagged and the dependents it has to follow, so proof bodies that already work are left alone. | Tagged Lean 4 source tree |
| F2.5–F4 | Proof and statement review loop | Checks first that the scaffold really does realize the frozen specification (F2.5). The proof-review loop then fills the remaining obligations and rechecks the frontier it changed against the live Lean 4 compiler (F3). Next come the deterministic unused-hypothesis lint and the scan for forbidden proof steps (F3.5), and last a full dual-model convergence review of the frozen graph (F4). The numbers F2.5, F3, F3.5, and F4 survive as labels for these substeps, but the work belongs to the loop. | Proof reviews, graph verdicts, crosswalk, and dependency ledger |
| F5 | Final approval preparation | Begins only once the scans for forbidden proof steps and the correspondence checks come back clean. It then writes a final TEX-to-Lean 4 crosswalk covering lemmas as well as theorems, updates the API documentation, and stops at the second checkpoint. Saving the run, committing, and promotion each need a human to say so. | Complete crosswalk and API update |

### E.1 TOPIC SELECTION

A run invoked with no topic starts with the topic-selection module, ahead of the numbered D stages. If a researcher supplies the topic instead, the module is skipped and the run enters D-1.1 directly. Figure 5 shows the flow.

The module first states a stance in one line. An exploit stance extends a past result, or works in a cluster where many attempts have been accepted. An explore stance goes to an area with few acceptances. The stance shapes how candidates are drafted, and every candidate is then ranked and reviewed on the same terms.

The module then searches recent work in the area and picks one or two anchor papers that carry a theorem. Subagents read each anchor in full, together with the papers that cite it. Each read returns

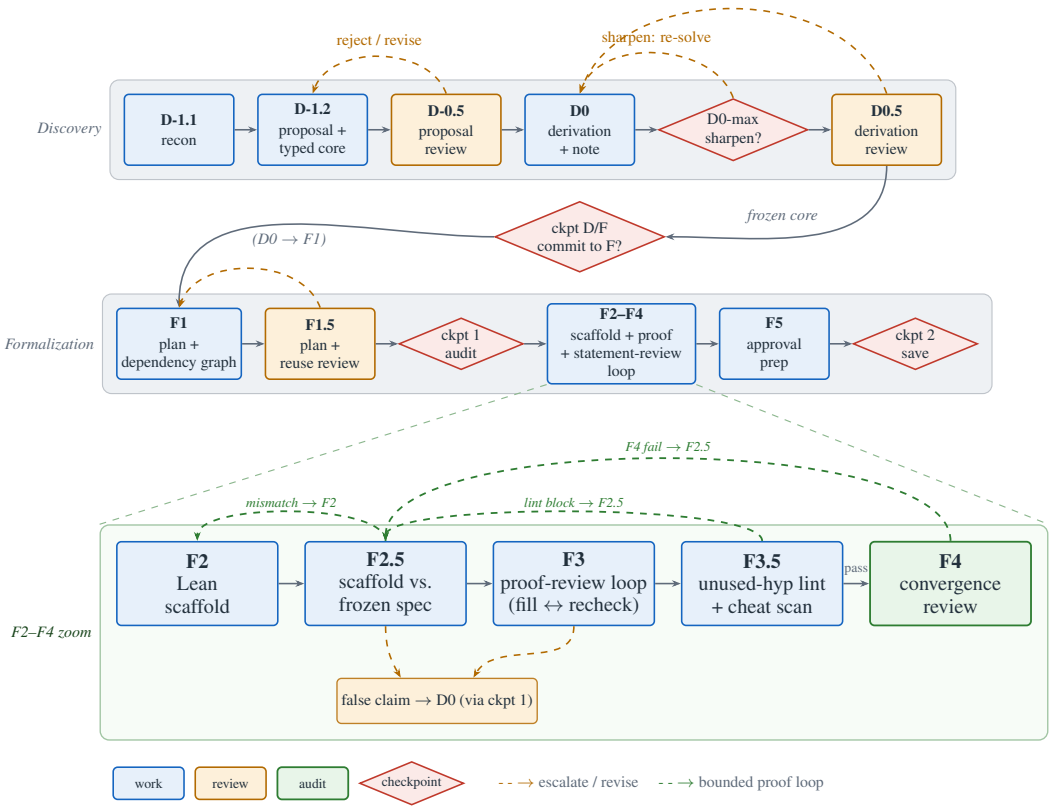



Figure 4: Stage flow of a CAUSALSMITH theorem run, with an inset zoom on the F2–F4 proof and statement-review loop. Top band: Discovery produces a proposal and a mathematical core, which D0.5 freezes and hands to Formalization (middle band). An amber review can reject the work or send it back one stage. The four red diamonds are halts that call for a judgment from outside the run. D0-max asks an oracle whether the result can be sharpened before D0.5 freezes it; ckpt D/F decides whether the result is worth the expense of F1–F5; ckpt 1 and ckpt 2 can ask a person to decide, before F2 and before the run enters the run history. Most runs used the automatic mode, where only ckpt 2 is normally held for a person. Bottom: the substages of the proof loop. A green arc sends work back for repair, while an amber arc escalates a false or under-specified claim to D0 instead of quietly rewriting the frozen statement. Table 6 gives the purpose and acceptance condition of each stage.

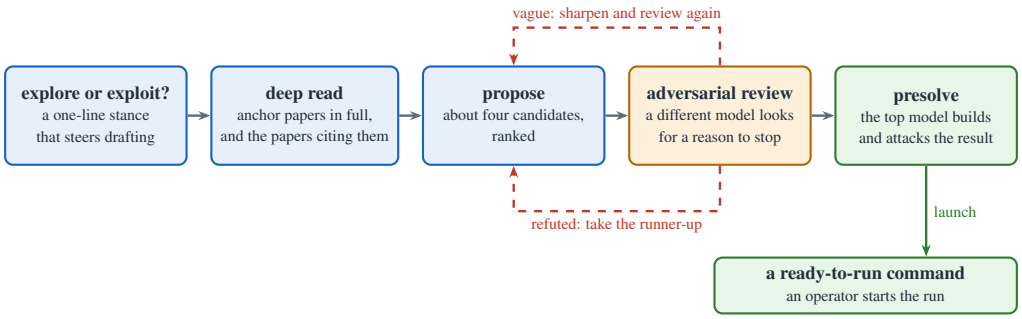



Figure 5: Topic selection. Blue boxes do the work, the amber box is the adversarial review, and red dashed arrows send a candidate back. Only a launch verdict from the presolve produces a command.

a short digest: the central theorems and their load-bearing proof steps, one or two questions left open at the boundary of the proof, and one or two directions where the same technique could reach a new target or setting. From these digests the module drafts about four candidates. It ranks them by

evidence, mathematical depth, novelty, feasibility, and value to a named consumer, that is, a published applied work or a software default whose practice would change if the result holds. Each candidate is also checked against the runs that are already active or finished.

The top candidate goes to an adversarial review run on a different model. The reviewer looks for a concrete reason to stop: a counterexample, a reduction to something trivial, or a paper that already has the result. It also asks whether the conjecture is stated precisely, whether the result would clear its target tier if true, and whether the consumer is real. A vague candidate comes back to be sharpened and is reviewed again, up to three times. A refuted candidate is dropped and the runner-up takes its place. If both fall, the module drafts one fresh slate, with the rejected topics and their failure modes as constraints.

A candidate that passes review goes to a presolve. In a single call the strongest available model attempts the result, attacks its own attempt with counterexamples and a fresh literature search, and writes the strongest supported theorem at full precision, with every gap listed. The call returns launch, pivot, or drop. Learning at this point that a result is out of reach costs one model call, while learning it during derivation costs a day of pipeline time. A launch verdict produces a ready-to-run command whose topic string carries a short summary of the presolve and the path to its draft, and an operator starts the run from it. Discovery treats the draft as evidence that still has to be checked.

### E.2 Proposal review

D-0.5 reviews the proposal against a rubric written in advance. A mechanical check has already confirmed that the proposal is well formed, so the reviewer spends its effort on meaning. Every finding carries one code from one of three families, and the family decides where the finding goes.

Structure findings check that the strongest result of the proposal is its headline target, instead of a lemma on the way to a weaker theorem. They lead to a structural patch.

Novelty findings are judged per statement on two axes, our own past runs and the published literature, and a statement counts as new only when it is new on both. A finding that a result is already known names the run or the paper. A finding that the proposal misstates a cited paper, or that the result is already published, has to come with a receipt giving the exact source version and a page, section, or equation. These findings send the proposal back for a redraft or a pivot.

Soundness findings ask whether each statement is well defined, whether it reduces correctly in the special cases it claims, and whether each assumption tagged as standard really is the standard condition under that name. At this stage soundness concerns the question itself; the truth of each statement is settled later, in D0. These findings lead to a redraft of the statement.

The review closes with a tier rating, scored against the target tier fixed when the run was launched. A repairable proposal returns to drafting a bounded number of times. A proposal that is already known, or too thin for its tier, is logged and dropped.

### E.3 Solving and derivation review

D0 solves the frozen proposal core, which already fixes the statements to prove and the dependencies between them. The solver groups the open statements into units (Figure 6), one per connected component of their dependency graph, so two open statements share a unit when a chain of open dependencies joins them. Each unit goes to its own agent, and the units run concurrently. An agent owns only the targets of its unit. It may add helper lemmas with their own proofs, and it declares every lemma and assumption its proofs rely on, so that the outputs of all units merge into one graph. A classical published theorem enters as a cited leaf, carried with its source and its exact statement. Each round regroups whatever is still open.

When a claim turns out to be too strong, the solver weakens it to something true and still interesting, or corrects the definition behind it. When a claim turns out to be too weak, the solver strengthens it. Either way the changed statement goes through the reviews again.

D0.5 splits the review of the derivation across separate referees (Figure 7). Correctness, fit with the rubric, and interest pull in different directions, and a single reviewer asked all three tends to pass a correct but dull result on correctness alone. The math reviewer re-derives each load-bearing step

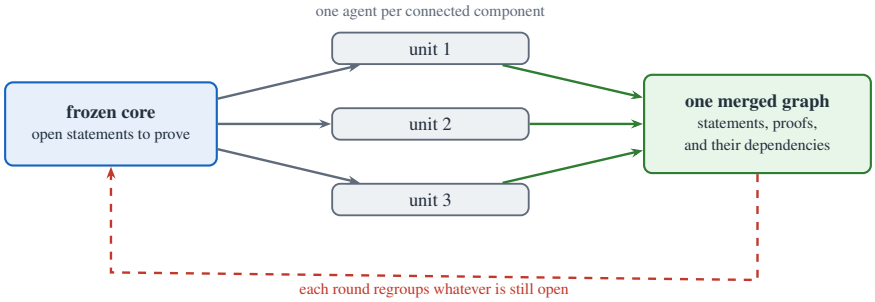



Figure 6: Solving at D0. The open statements are split by the connected components of their dependency graph, the units are solved concurrently, and their outputs merge back into one graph.

from the definitions, then compares its own derivation with the note term by term and reports every difference. The rubric reviewer checks structure, novelty, and positioning against the written rubric. When the target tier calls for it, a general referee reads the note cold. It sees the note and plain tier definitions and nothing else. The solver itself works toward the rubric, so a note can satisfy every item and still fall short, and the general referee is kept away from the rubric for that reason. It grades what is proved over what is promised, and it calibrates against a spread of accepted papers together with the scores their final referee gave them.

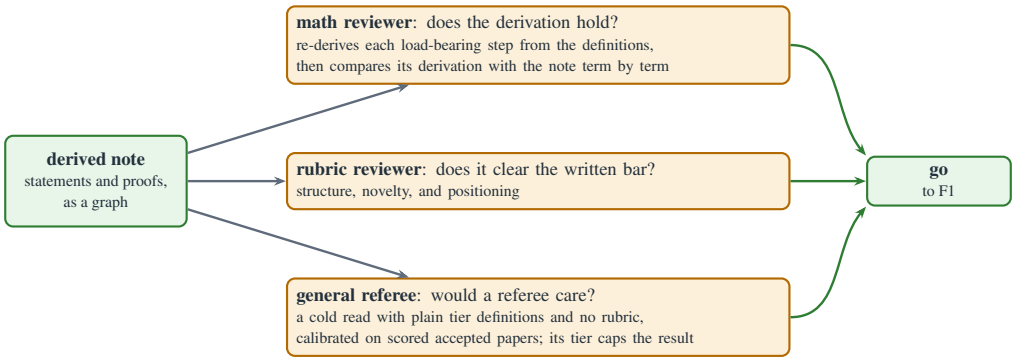



Figure 7: The derivation review at D0.5. Each referee answers one question. The general referee joins when the target tier calls for it. A math failure, or a salvageable result below the target tier, sends the run back to D0.

A failure from the math reviewer sends the run back to D0. A general referee that places the result below the target tier, but judges it salvageable, sends the run back to D0 with a directed improvement, at most twice by default. A result judged below the target tier and beyond repair halts the run for a decision.

## F How the graph drives formalization, proof, and audit

The pipeline is built around the graph. The D stages create it in natural language. The typed core of D-1.2 names the objects, assumptions, and statements together with the edges between them, D0 extends it while deriving the result, and D0.5 freezes the argument, one node per claim. The F stages attach Lean 4 to it. F1 gives every node an intended declaration and module, F2 emits the scaffold, and each pass of the proof loop reads the emitted declarations and dependencies back into the graph. Formalization also adds nodes for the intermediate lemmas the derivation never had to name. The graph thus maps each planned object to what it should become, and each emitted declaration to what it realizes.

At F1.5, code checks what has a definite answer: whether each cited declaration exists, whether every node is covered, and whether the dependencies close. The model judges the rest. It asks whether

a reused lemma sits at the right level of abstraction, whether the plan searched the library before rebuilding, and whether a missing fact should be built or carried as an openly disclosed assumption.

The statement audit runs at F2.5, F3.5, and F4; a second audit, in Presentation, checks the paper against the frozen graph (Section G). Each node is `unreviewed`, `matched`, or `drift`. An accepted review is stamped with a fingerprint of the content it reviewed, so a change to a node or its realization sends that node and its downstream nodes back to review.

In the statement phase, the reviewer compares each frozen statement and definition with the Lean 4 scaffold. Typical drift is a dropped hypothesis, a constant fixed where the claim was general, or a definition narrowed until the claim becomes easy. A mechanical mismatch goes back to F2 for a local revision. A false or under-specified claim, or a missing library fact, is escalated. The frozen natural-language statement stays as written throughout, since it is the contract the audit protects.

Proof filling starts once the statement phase is clean. One prover session runs at a time, because the statements of one result sit in a single import chain, and each session starts from a summary of the earlier ones. Each iteration reviews the frontier it changed, attempts the open obligations against the live compiler, refreshes the graph, and checks for progress. The loop finishes with a successful build, no `sorry` or `admit`, the frozen theorems and their dependency closures proved and matched, a clean scan for `axiom`, `opaque`, and `unsafe`, and a passing unused-hypothesis lint. An unused hypothesis suggests that the assumption is decoration or that the statement says less than intended. A blocking finding returns the loop to the statement phase, and the rest are logged for the orchestrator.

The final convergence review runs independently of the incremental one, on GPT and on Opus. It revisits every object that originates in the paper, each theorem, definition, lemma, and symbol with its assumptions, and judges helper lemmas through the statements that use them. Two different models share fewer blind spots than one model asked twice. When they disagree, the orchestrator reproduces the complaint. A correct complaint is fixed in the code, and a mistaken one becomes a general fix to the reviewer prompt. The review also notes how each library dependency was treated. A `gated` dependency appears as an explicit conditional assumption or as a visible build obligation. A `cited` dependency is checked against its source and blocks approval if its encoding is mismatched or underspecified. F5 then writes the complete crosswalk, updates the API documentation with a doc comment on every public declaration, and marks the run as accepted, downgraded, or failed.

## G PRESENTATION PIPELINE

The theorem pipeline and the paper pipeline are separate state machines. The paper pipeline takes an accepted result and runs stages P0 through P5, with an optional P6 for slides (Figure 8). The paper is generated from the checked code, and every later pass asks whether the prose has drifted from it.

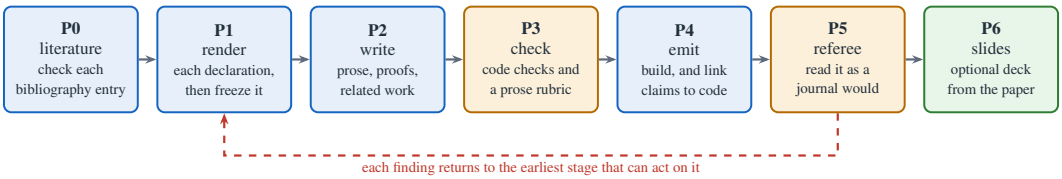



Figure 8: Stages of the presentation pipeline. Blue boxes produce the paper, amber boxes review it, and the red dashed arrow is the revision route taken after the referee review.

P0 assembles the literature. A model proposes the bibliography, and code looks up each entry in Crossref, arXiv, and OpenAlex and drops any entry that fails to resolve. Every citation in the paper has to come from this resolved pool.

P1 builds the formal layer of the paper. It walks the graph and renders each verified declaration as a displayed theorem, definition, or assumption in ordinary mathematical notation, ordered so that every object appears before anything that uses it. Each rendered statement is judged against its Lean 4 declaration at the moment it is rendered. The English statement and its Lean 4 counterpart have to be logically equivalent in assumptions and in conclusion alike. If either one is the stronger, the pipeline halts for adjudication and keeps the declaration it was aiming at. The accepted layer is then frozen.

Later passes rewrite the prose around it freely, and every frozen statement is restored byte for byte afterwards.

P2 writes everything else: the introduction and abstract, the motivation, the proof of each result, and the comparison with related work. The prose that describes proofs is audited against the graph. When that audit fails, the pipeline runs one promotion round. It adds the statements the proofs need to the formal layer, renders them at P1, and retries P2 once.

P3 reads the assembled draft as one document. Code checks first that every citation key lies in the resolved pool, that every cross-reference resolves, and that the statements stand in the order P1 fixed. A model then asks whether any sentence claims more than the theorems prove, whether each cited source supports the sentence that cites it, and how the draft scores on a rubric covering claims, positioning, assumptions, and writing. Revisions touch the prose alone, and the draft gets at most two rounds.

P4 emits the rendered paper with its links to the code. Links are assembled from the graph first. For each object it displays, the presentation system pulls the relevant declaration and its `statement-uses` neighbors out of the verified graph, emits a source anchor, and checks that the dependencies and assumptions on display have links of their own. A link therefore resolves to an object that has passed the statement audit. P4 also ties each prose statement to its declaration hypothesis by hypothesis, which drives the crosswalk of Section I.

P5 reads the paper as a journal referee would, taking the theorems as given. It asks whether the contribution is significant and has been achieved, whether each prose claim matches the verified statement in strength and scope, whether the paper engages the actual results of its closest competitors, and whether each condition is stated where it is claimed. Each finding is routed to the earliest stage that can act on it. The orchestrator then revises the sources by hand and re-enters the pipeline at that stage, after which the downstream reviews run again. P1 and P2 double as outline and draft checkpoints: resuming from either one approves going on, and re-entering explicitly at P0, P1, or P2 allows a targeted revision. Once the paper settles, P6 can build the slide deck, taking every statement verbatim from the frozen layer.

## H STUDY MODE

Partway through a proof, formalization can reach a step that needs a lemma nobody has written. Three responses are reasonable. The run can note the gap and ship as an openly conditional result, leaving the lemma for a later run. It can prove the lemma in place, next to the paper, which is the quickest route to a finished result. Or it can build the lemma for the library, stated at its natural generality, so that later runs find it ready. The choice turns on how wide the gap is and how many results are likely to want the same fact. The orchestrator sends a fact of general use to study mode, and builds a fact that is specific to one model and of manageable size in a helper file inside the run. The theorem run carries on with the fact assumed. Once the lemma lands, the new declaration returns to F1 as a candidate for reuse, and the statements that use it are reviewed again.

A study starts from a short requirement written in plain English (Figure 9). It states the fact that is needed, the generality it should have, and how the literature states it. The requirement leaves the home of the code open. A scaffolder plans the lemmas and their order and writes a file with every statement present and every proof left open. A prover then fills the proofs against the live compiler. Provers run one at a time for the same reason as in F3, since the new files usually form one import chain. After each round the scaffolder reads the build report and decides whether to run another round or send the module to review, with at most ten build rounds.

A reviewer then judges the module against seven checks, all of which have to hold. The module is generic: it states the general fact, at the level where the fact holds. It is reusable: a clean, well-named statement whose hypotheses are the ones a later proof would reach for. It is standard: it matches how the literature states the result. It is non-vacuous: its hypotheses are satisfiable by real inputs, and the statement keeps its full strength on the hard case. It fulfills the requirement, with sharp constants intact; the reviewer judges the requirement met and accepts a proof route that differs from the one the requirement sketched. It is free of `sorry`. And it is layered: its imports stay within Mathlib, CAUSALEAN, and the files of the study itself. The last check protects the direction of dependency, since a library lemma that imports a research file would carry that paper into the library for good.

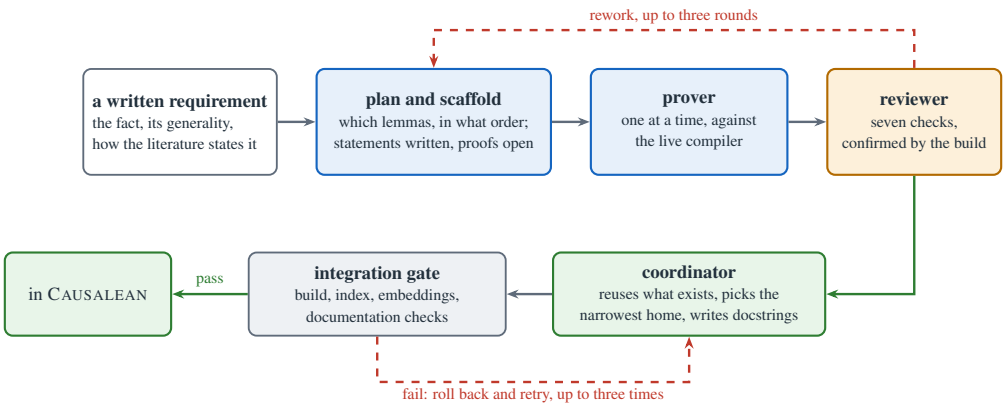



Figure 9: A study-mode build. The top row writes and proves the module; the bottom row places it in CAUSALEAN. Red dashed arrows are the bounded rework routes.

The build confirms a passing verdict on its own, so a module that fails to compile or still has an open proof goes back to the scaffolder. The review allows three rounds.

A coordinator then places the module. It searches CAUSALEAN for declarations that already prove the same statement and reuses them. It picks the narrowest existing file or subject area that fits, and merges into an existing file where it can. Its edits are additive, consisting of new files and inserted blocks, so every existing line of the library stays as it was. It also writes the docstrings that the library explorer and the retrieval index read. Code then runs the integration gate: a full build, re-indexing, the retrieval embeddings, and the documentation checks. A failure rolls CAUSALEAN back to its prior state, and the coordinator retries with the failure log, up to three times. A gate that hangs stops for the orchestrator, since rolling back in the middle of a build is unsafe. Concurrent studies take turns at this step, because each one edits the shared library. Once the gate passes, the lemma is part of CAUSALEAN. The library therefore grows out of what the pipeline demands, with a boundary one can audit between a result specific to a single run and a theorem the library will reuse.

# I THE READING INTERFACE

A finished run leaves three things behind: a paper, a Lean 4 development, and the frozen graph that ties them together. The companion site, whose link is withheld for review, presents all three on one page. It adds four things on top of the text. It draws the dependency structure of the paper's own proofs. It puts each English statement next to the Lean 4 declaration that was audited against it. It lets a reader record what they checked and what they thought of it. And it presents the same result as a short 20-minute talk. The screenshots in this section all come from one accepted run, "Sharp Minimax Rates for Average Treatment Effects with Discrete Confounding under Fixed Overlap", and show the site as it stands. Reader handles and avatars, outbound repository links, and pinned commit hashes are blurred for anonymous review; nothing else in the screenshots is altered.

## I.1 THE PAPER PAGE

Figure 10 shows a paper page. The middle column is the paper itself. The left rail holds the table of contents and the proof map. The right rail holds reader comments, each one beside the passage it is about. The byline carries the pipeline's own reviewer score, the reader rating, the commit version number the page is pinned to, and links to the PDF, the Lean 4 development, and the slides.

## I.2 PROOF MAP

The proof map (Figure 11) is the reader-facing view of the frozen graph of Figure 3, restricted to the results the paper displays. Each node is a theorem, proposition, or lemma of the paper. An edge means that one proof invokes the other. The layout is layered: a result sits one rank above the deepest




Figure 10: A paper page. Contents and proof map on the left, the paper in the middle, reader comments on the right. The byline gives the reviewer score, the reader rating (four of five stars from two readers), the pinned commit, and links to the PDF, the Lean 4 code, and the slides.

result its proof invokes, and the theorems are pulled to the top row, so the paper's headline results form the roof of the structure. The paper shown here has 22 statements and 29 such edges.

Selecting a node highlights the results its proof invokes and the results that invoke it. It also fills a card underneath with the statement, both lists as links, and the reader marks on that statement. Every node is also a link, so a reader can go straight from the map to the statement it names.

### I.3 Natural-language statement and Lean 4 declaration

Every statement in the paper is clickable, and clicking it opens the Lean 4 side of it (Figure 12). The panel gives the declaration name, its file and line, whether a scan of that source finds it free of `sorry`, and the statement itself, split into its hypotheses and its conclusions. Where a symbol in the paper is realized by a Lean 4 definition, that definition is shown too, with a link back to the place in the paper that introduces it. Two kinds of gap carry an explicit label: a hypothesis the Lean 4 statement holds that the paper leaves unstated, and a declaration in the development that no paper statement cites.

The pipeline claims that the English statement and the Lean 4 one express the same proposition. The crosswalk puts the two side by side, so a reader can make the same comparison and disagree with it. A separate page lists the complete development for the paper, module by module, including the helper lemmas the paper never cites.

### I.4 Reader Interaction

A reader signed in with a GitHub account can leave three kinds of mark. All are public and all are attributed.

The first is a verification of a single statement, in the site's own words: I read this statement and its proof and found them correct, taking the results the proof invokes as given. The unit is deliberately one statement. The results a proof invokes are somebody else's to check, so the work is small enough that a reader can finish it in one sitting. The proof map shows how far this has gone, both as a count in its header and as a dot on each node. The paper in Figure 11 has 12 of its 22 statements verified.

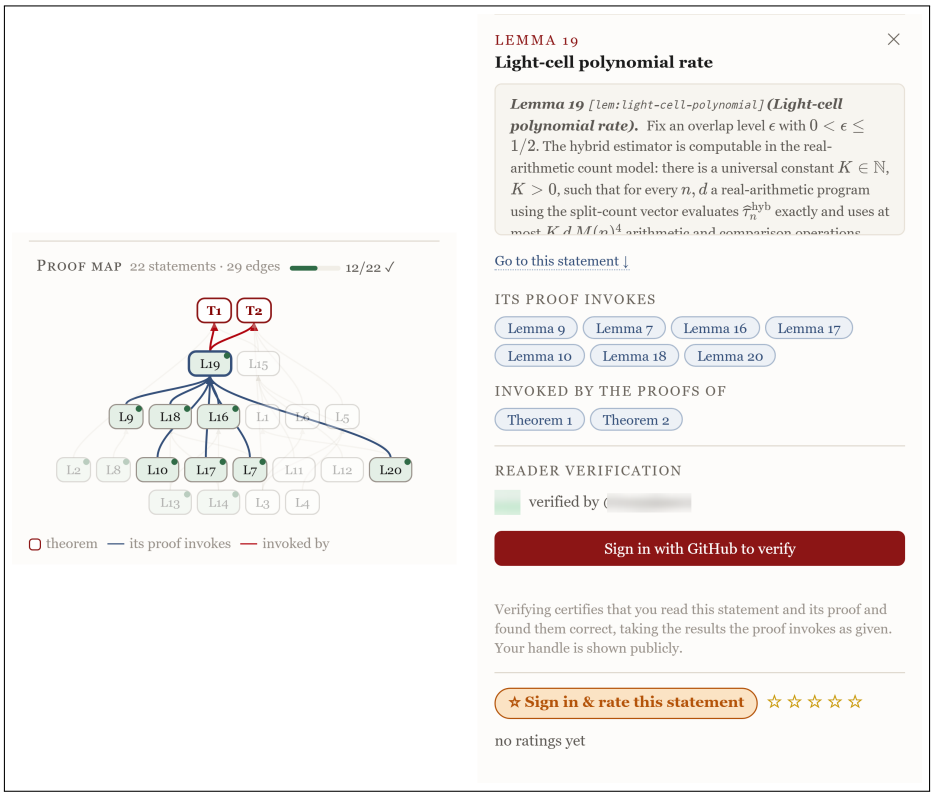



Figure 11: The proof map, with one lemma selected. Left: the map itself, with the selected node's dependencies drawn in and the rest of the graph faded. Blue edges run to the results the selected proof invokes; red edges run to the two theorems whose proofs invoke it. Right: the card the selection fills, giving the statement, both dependency lists as links, and the reader marks. The header counts how many of the 22 statements a reader has verified.

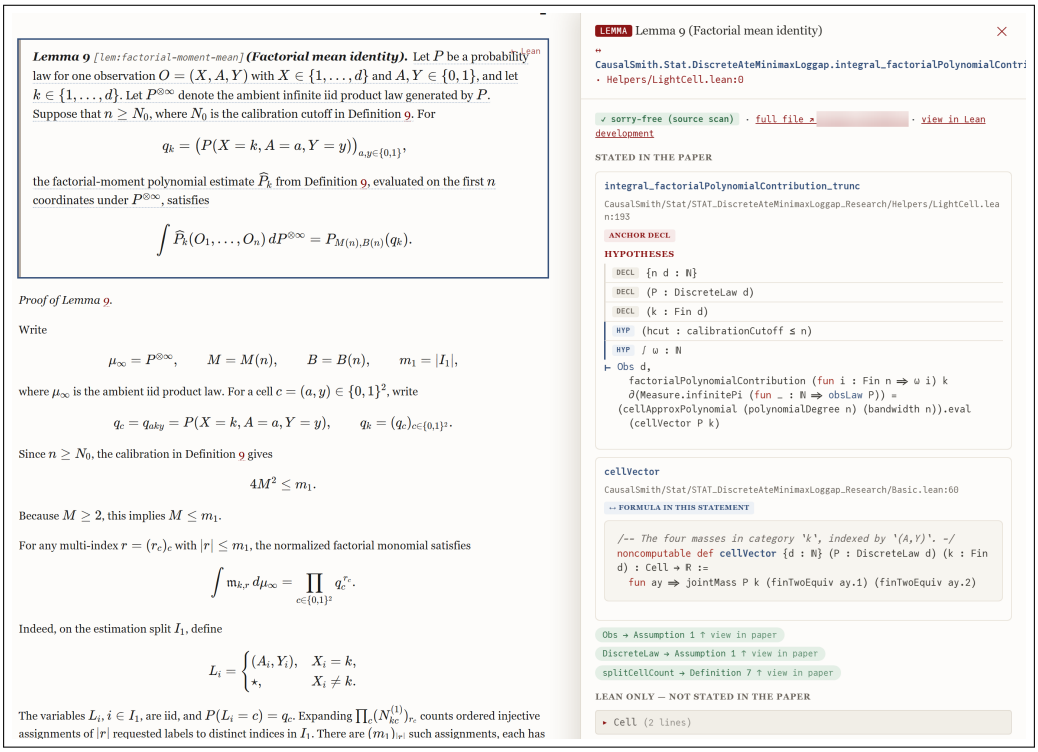



Figure 12: A statement in the paper (left, outlined) next to its Lean 4 declaration (right). The panel gives the source location, the `sorry` scan, the hypotheses and the conclusion, and the definitions the statement rests on, each linked back to the paper. The last rows list declarations the Lean 4 development uses and the paper leaves unstated.

The second is a rating of one to five stars, on the paper as a whole or on a single statement, one per account and withdrawable by clicking the same star again. The page shows the average and the count. This sits beside the pipeline's own reviewer score and answers a different question. The score is one model's judgment at the end of the run. The rating is what readers thought afterwards. The paper of Figure 10 has four of five stars from two ratings.

Selecting any passage of the paper offers to comment on it, and the comment is filed in the margin beside the passage it quotes (Figure 13). A comment can be left plain, or tagged as a verification or as a problem, and replies thread underneath it.

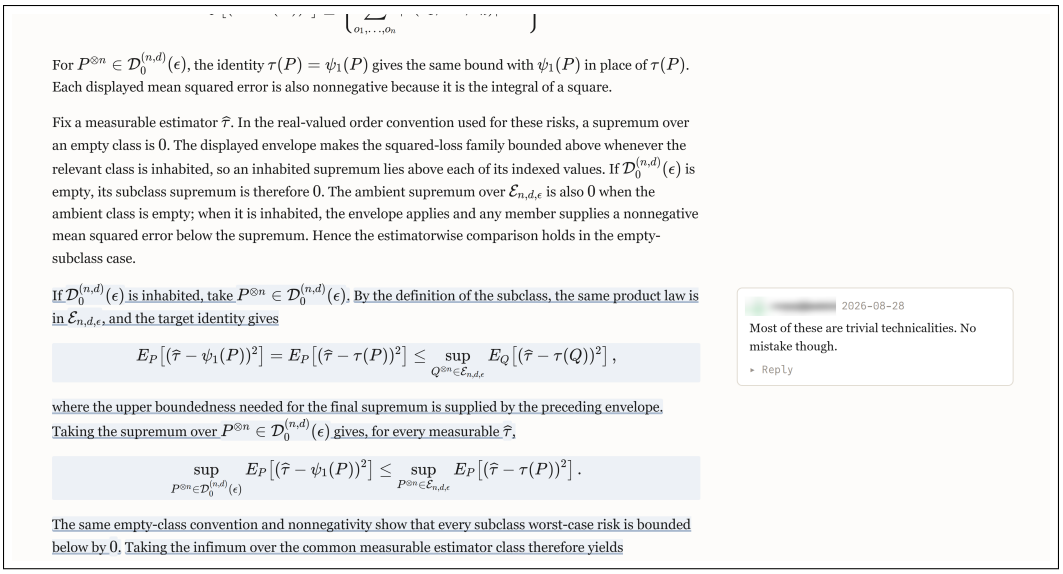



Figure 13: A reader comment. The quoted passage is shaded in the proof and the comment sits beside it in the margin, with its author, its date, and a reply thread.

### I.5 SLIDES

Each accepted result also ships as a short deck, built from the same graph as the paper (Figure 14). The deck for this paper runs to 17 slides: the setting, the literature position, the idea, the main result, a proof sketch, and what follows.

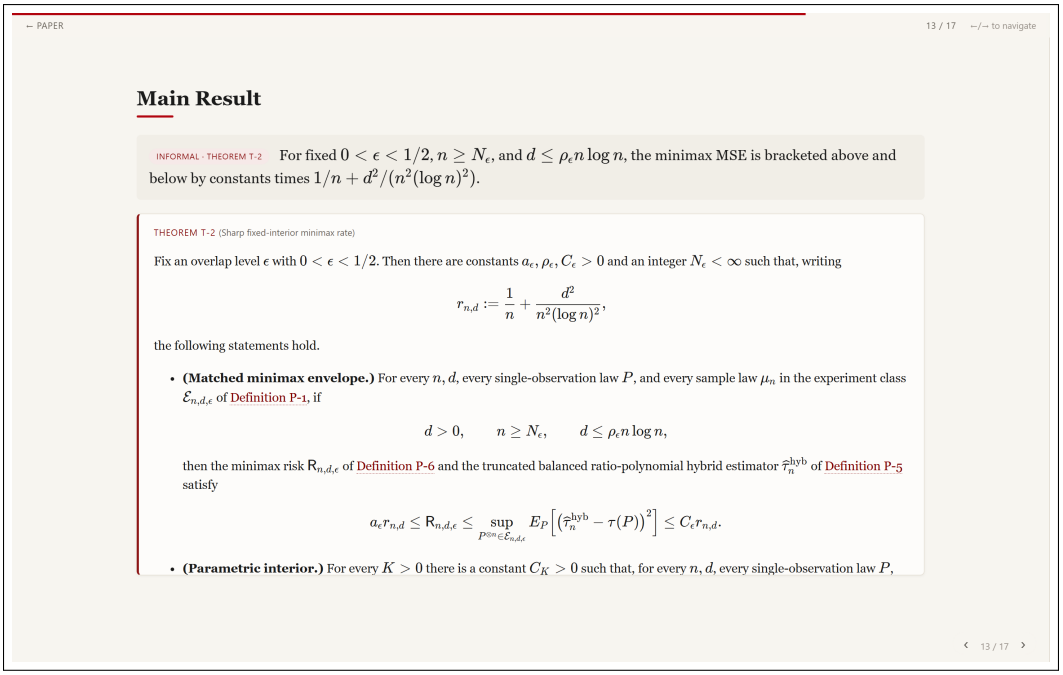



Figure 14: A slide carrying the main result. The one-sentence restatement at the top is labeled informal, and the audited statement follows below it, with its definitions linked back to the paper.

# J REPRODUCIBILITY

The library and the pipeline are both pinned to Lean 4 toolchain `leanprover/lean4:v4.29.0-rc3`. We read the declaration counts of Table 5 from the compiled environment index, which `lake exe library_index` prints. The axiom checks of Section 6.3 come from running `#print axioms` on each headline theorem in a clean build, and we cross-check them with a comment-aware scan for `sorry`, `admit`, and `axiom` over every run module and the library closure it imports.

The anonymous replication package submitted with this paper contains the pinned Lean 4 projects, the agent implementation and stage prompts, the labeled 407-run campaign manifest, and the redacted run history; its `evaluation/reproduce_tables.mjs` script reconstructs Table 2 with no network access and no model calls. The companion site, whose link is withheld for review, gives a browsable view of the library and of the accepted results. The run catalog behind Table 2 is the set of banked run directories under `CausalSmith/doc/research/_bank/`, one directory per run, filed under `accepted/`, `downgraded/`, or `failed/`. Each directory carries the run's artifacts verbatim, that is, the state file, the proposal, the review log, and the derivation note, and it is from these that we read the dispositions and the downgrade reasons we quote. Legacy runs, meaning the ones from before the current pipeline, are left out of both cuts of the catalog.

The cluster labels in Table 2 are CAUSALSMITH's own six clusters. Each run's state file names its cluster, and the run identifier carries it as a prefix (`stat_`, `exp_`, `panel_`, `eid_`, `pid_`, `scm_`). Discovery assigns the cluster at proposal time and picks the cluster-specific setup prompt from it, so the grouping is fixed before anyone knows the run's mathematics or how it will turn out. The question-type labels are a different matter, since the pipeline does not produce them. We assigned them ourselves, from each run's proposal-time `topic` string, by the following rule.

- **Method comparison.** The topic compares two or more published estimators or identification frameworks and asks how they differ: non-equivalence, strict-extension, or equivalence.
- **Assumption relaxation.** The topic weakens a named assumption, under a budget on how far it may be violated, and asks for the identification result—an identified set, a threshold, a phase transition—as a function of that budget. Sensitivity and ambiguity models are the common instances.
- **Gap filling.** The setting already exists in the literature, but some gaps are left open: a matching bound; an exact rate, order, or constant; a minimax optimum or optimal design; or an explicit identifying formula.
- **New setting.** Anything else: a newly posed object, criterion, algorithm, or framework, with no published counterpart to be matched against.

That string is the `topic` field in the banked run's `README.md`, and it too is fixed before the run's mathematics, so a reader who would draw the boundaries elsewhere can apply their own rule to the same text. For the stated-versus-inferred provenance reported in Section 6.1 we read the anchor paper of each accepted run; we did not repeat that reading for the downgraded and failed runs.

Two parts of the library are not original to CAUSALEAN. Both are adapted copies of external Lean 4 projects, bumped to CAUSALEAN's Lean 4/Mathlib pin. The first is the Karush–Kuhn–Tucker first-order necessary conditions under LICQ and affine constraint qualifications, taken from OptSuite's `optlib` (Li et al., 2025) and consumed by the Estimation cluster's minimax lower bounds. The second collects Rademacher complexity, McDiarmid's inequality, symmetrization, and the Dudley entropy integral, taken from `lean-rademacher` (Sonoda et al., 2025) and consumed across Stat, Estimation, and ML.

**Base models.** Every pipeline stage dispatches to one of two agent runners, OpenAI `codex` or Anthropic `claude`. Each logical role maps to a model id that is committed in the pipeline source and can be overridden per role by an environment variable (`CAUSALEAN_MODEL_*`). By the close of the campaign the assignment stood as follows. GPT-5.6 Sol held the hard-mathematics tier, which is the proposal, the derivation, the referee review, and the Lean 4 proof filler, and it also served as the `codex` half of the dual convergence review. GPT-5.6 Terra held the mechanical tier: scaffold translation, banking, proposal screening, and artifact emission. GPT-5.5 drafted and revised the

presentation. Claude did the planning, the Lean 4 code review, the second convergence review, and the presentation rubric scoring.

Both sides were updated during the campaign. On the OpenAI side the hard-mathematics tier moved from GPT-5.5 to GPT-5.6 Sol on 2026-07-10, and for the day before that the proof-filler role sat on the mechanical tier as well. On the Anthropic side the Claude roles ran as Opus 4.7 until 2026-05-28, as Opus 4.8 until 2026-07-24, and as Opus 5 thereafter. Our campaign ran from 2026-05-14 to 2026-09-16, so cut at those two dates the $407$ runs of Table 2 divide into $81$ under Opus 4.7, $42$ under Opus 4.8, and $284$ under Opus 5, with none spanning a change; of the twenty-two accepted runs, eight sit entirely under Opus 4.8 and the other fourteen entirely under Opus 5.