LIBRARY CARD: AUTOMATED REPOSITORY KNOWLEDGE INTERFACE FOR LEAN THEOREM PROVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated theorem proving (ATP) in large Lean repositories requires both retrieving relevant premises and providing repository-specific information that supports their effective use. Premise retrieval is central to repository-aware theorem proving, but premise availability and proof-time context are often evaluated jointly. To address this, we introduce Library Card, an automated repository knowledge interface for Lean theorem proving. Library Card automatically constructs and organizes repository-specific knowledge, then uses it to independently control the premises available for selection and the context provided around selected premises. This separation enables a controlled study of whether proving performance is limited by finding additional premises or by using available repository knowledge more effectively. On a 50-theorem benchmark from Optlib, the baseline formal-plus-semantic retriever recovers all 37 observable earlier-same-file source-proof premise occurrences at $K = 50$, while Card-based expansion rarely changes the selected premise set. With selected premises and their order fixed, the observed solve rate is 54% with rich context versus 44% with compact context, although the paired difference is statistically inconclusive. Additional controls show no aggregate advantage for the tested structured or compiled representations. These results suggest that when the evaluated retrieval budget already covers the observable premises in scope, progress may depend less on expanding candidate availability and more on how repository-specific knowledge is provided to the prover. Our source code and data are available at `https://anonymous.4open.science/r/Library-Card`.

1 INTRODUCTION

Formal theorem proving provides machine-checkable guarantees for mathematical reasoning, but scaling mathematical formalization remains challenging because proofs must be expressed in a precise formal language and verified by a proof assistant (Moura & Ullrich, 2021; Zhou et al., 2026). Recent language-model-based provers have substantially improved automated proof generation through stronger reasoning, verifier feedback, and search (Lin et al., 2026; Varambally et al., 2026; Xin et al., 2025). As automated theorem proving (ATP) moves from isolated benchmarks to repository-scale settings, successful proof construction increasingly depends on project-specific definitions, dependencies, and previously established results (Kumarappan et al., 2025; Xin et al., 2026). A central challenge is therefore to identify and effectively use goal-relevant knowledge from a large formal library (Yang et al., 2023; Blaauwbroek et al., 2024). Recent premise-selection methods further show that retrieving the right declarations is critical for automated proving (Zhu et al., 2026; Gao et al., 2026), but access to relevant premises alone does not determine how effectively a prover can use them (Hu et al., 2025; Thompson et al., 2024). This raises a natural question of whether improvements in proof generation come from retrieving additional relevant premises or from providing richer repository-specific information about premises that are already available.

Prior work has incorporated repository information into theorem proving through premise retrieval, contextual guidance, and repository-level adaptation. ReProver retrieves relevant declarations for Lean proof generation (Yang et al., 2023), while miniCTX and Rango provide broader project context through surrounding file content, retrieved premises, or similar proofs (Hu et al., 2025; Thompson et al., 2024). LeanAgent further uses repository data to adapt retrieval across mathematical projects (Kumarappan et al., 2025). Together, these systems demonstrate that repository knowledge

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.

can influence multiple stages of the proving pipeline. Yet existing evaluations often couple changes in which premises are available for selection with changes in the information provided around selected premises, making their respective contributions difficult to distinguish. This motivates studying premise availability and proof-time repository context as separate interventions.

To disentangle the effects of premise availability and repository context, we propose Library Card, an automated repository knowledge interface for Lean theorem proving. Library Card constructs and organizes repository-specific records covering declarations, concepts, theorem families, and proof patterns. It then exposes this knowledge at two separate stages of the proving pipeline. Before premise selection, repository links can expand the candidate pool. After premise selection, the same ordered premises can be presented either as compact formal statements or with additional semantic, structural, and verified-use information. The prover parameters remain fixed throughout, allowing changes in candidate availability and repository context to be studied without confounding them with model adaptation.

To study these effects under controlled conditions, we construct Optlib-50, a benchmark of 50 deterministically selected theorem targets from Optlib, a Lean library for formalized optimization theory (Li et al., 2024). At the evaluated same-file cutoff of $K = 50$, the baseline retriever recovers all 37 observable source-proof occurrences in scope before final selection, while the shared Top-10 selector retains 30 of them. Card-based candidate expansion changes the final selected premise set for only 2 of 50 targets. In contrast, when the selected premises and their order are held fixed, richer repository context changes the observed solve rate from 44% to 54%. The paired difference is not statistically conclusive, and additional representation controls do not identify a consistent advantage for the tested structured or compiled forms. Together, these results indicate that candidate discovery, final premise selection, and repository context should be evaluated as distinct stages rather than treated as a single retrieval problem.

Our contributions can be summarized as follows: (1) We introduce Library Card, an automated repository knowledge interface that constructs and organizes repository-specific information for Lean theorem proving. (2) We disentangle premise availability from repository context, enabling their effects to be controlled and evaluated independently under a fixed proving pipeline. (3) We provide a stagewise empirical analysis of candidate discovery, final premise selection, and proof outcomes, showing that the evaluated $K = 50$ retrieval setup covers the observable premises in scope while repository context remains a meaningful source of variation in proof behavior.

2 Related Work

Repository context and external knowledge. Repository-aware systems provide information beyond the current proof state through source context, retrieved examples, and external knowledge interfaces. For source context, miniCTX evaluates Lean proving with preceding file content, while VeriSoftBench compares full-repository inputs with curated context derived from reference-proof dependencies (Hu et al., 2025; Xin et al., 2026). For example-based guidance, Rango retrieves premises and similar proofs from the current Coq project (Thompson et al., 2024). In the related task of statement autoformalization, DRIFT retrieves illustrative theorems to demonstrate how retrieved premises are used (Zhang et al., 2026). For knowledge access and integration, LeanExplore organizes declarations, informal descriptions, and dependency information for search (Asher, 2025), LAMP exposes a curated domain ontology through tools (Santhana Srinivasan R & Patawar, 2026), and Lean Copilot integrates model inference, premise selection, and proof automation into Lean (Song et al., 2024). Library Card combines automated repository-record construction with controlled use of those records before and after premise selection. Its fixed-premise comparisons examine the effect of accompanying repository information without changing the selected declarations or their order.

Premise retrieval and structural retrieval. Premise-selection methods identify useful library declarations through learned relevance scores, semantic search, and structural representations, with additional mechanisms for handling newly available knowledge. In learned and semantic retrieval, ReProver and Magnushammer learn premise selection in Lean and Isabelle, respectively (Yang et al., 2023; Mikuła et al., 2024). LeanSearch supports informal queries over Mathlib, Lean Finder aligns retrieval with user intents, and LeanSearch v2 extends retrieval to theorem-level premise sets through iterative proof-sketch-guided search (Gao et al., 2024; Lu et al., 2026; Gao et al., 2026). In

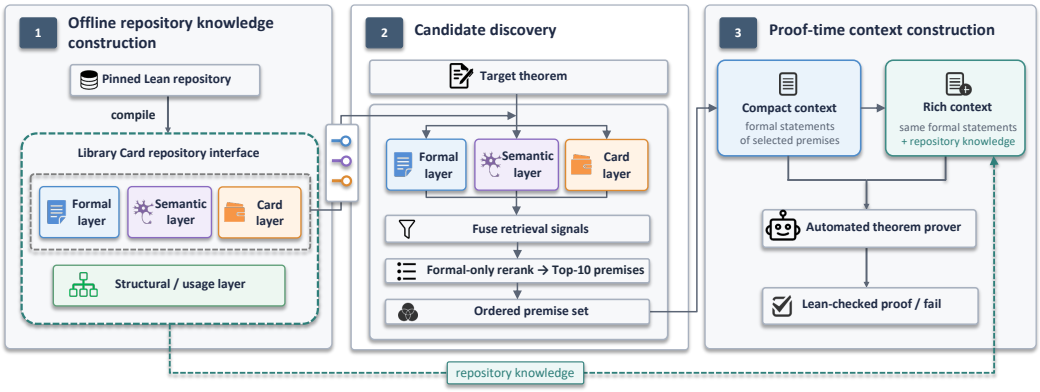

Figure 1: Repository information is compiled offline. Candidate discovery selects premises before compact or rich context is rendered for the frozen prover; Lean checks the final proof.

structural modeling, graph embeddings encode logical formulas for premise selection, while later Lean methods incorporate declaration-dependency graphs (Wang et al., 2017; Petrovčič et al., 2025). Graph2Tac constructs representations for new Coq definitions, LeanPremise incorporates local and out-of-training-library premises, and LeanAgent progressively trains its retriever across repositories (Blaauwbroek et al., 2024; Zhu et al., 2026; Kumarappan et al., 2025). Library Card retains a shared final selector and analyzes candidate coverage separately from the premises retained after selection. Its context intervention then varies the information accompanying a fixed premise selection.

Inference-time knowledge access and context use. Inference-time theorem-proving systems incorporate auxiliary knowledge through explicit knowledge interfaces, retrieval-and-feedback pipelines, and contextual conditioning. LAMP provides Lean agents with tool-based access to structured domain knowledge, while Lean Copilot integrates model inference, premise selection, and proof automation within Lean (Santhana Srinivasan R & Patawar, 2026; Song et al., 2025). COPRA and Hilbert combine retrieved information with proof states, decomposition, and verifier feedback during inference (Thakur et al., 2023; Varambally et al., 2026), while Goedel-Architect organizes proof construction through iteratively refined proof blueprints (Chung et al., 2026). More broadly, long-context studies show that additional context is not always used reliably by language models (Liu et al., 2024; Hsieh et al., 2024). Library Card instead keeps the prover and execution protocol fixed, and studies how repository-specific information is organized and exposed before and after premise selection.

3 Library Card

Figure 1 gives an overview of Library Card, which is organized into three stages. First, an offline repository-compilation stage constructs a reusable repository knowledge interface containing formal, semantic, Card-based, and structural/usage information. Second, at proof time, the candidate-discovery stage uses the target theorem to retrieve and fuse repository signals, then applies a shared formal-only selector to produce an ordered set of premises. Third, the proof-time context stage renders those selected premises either compactly, using only formal statements, or richly, with additional target-available repository knowledge. This separation allows Library Card to use the same compiled repository information in two distinct ways—to influence which declarations are available before premise selection and to influence how already selected premises are presented afterward—while keeping the prover and Lean verification pipeline fixed.

3.1 Repository records

The pinned repository is compiled into canonical declarations, Lean signatures, source provenance, deterministic dependency, concept, and theorem-family relations, together with repository-use rela-

tions used later to construct verified-use evidence. A deterministic heuristic builds 1,518 Cards of four kinds: declaration, concept, theorem-family, and strategy. Declaration Cards organize repository information around individual Lean declarations; concept and theorem-family Cards group related declarations, while strategy Cards capture recurring proof-use patterns. All Card types ultimately expose links back to concrete repository declarations, allowing retrieved Cards to expand the declaration candidate pool.

A separate semantic compiler produces structured, evidence-anchored descriptions from formal signatures and repository relations. Semantic records and Cards are parallel offline products rather than one being generated from the other; both retain provenance needed for target-time filtering.

A genuine Card (abridged). The declaration Card below is taken from the stored corpus; fields are shortened only for display.

Name: `ADMM_Converge_Proof.hconverge` **Kind:** lemma
Formal result: the ADMM subsequence triple (x_1, x_2, y) tends to the product neighborhood of (x_1'', x_2'', y'').
Direct dependency: `admm_nhds_prod_eq`.
Goal pattern: apply a known lemma to close or simplify a goal.
Known users: the x_1, x_2, and y subsequence-convergence lemmas, and `xy_isBounded`.

3.2 Candidate discovery

For a target, *legal* means available before that declaration: candidates come from strictly earlier declarations in the same source file, excluding the target and same-formal-type duplicates. Other Optlib files and imported Mathlib declarations are outside this retrieval task, although imports remain available to Lean.

The formal channel ranks legal declarations by their formal-statement representation, while the semantic channel ranks the same declarations through their accepted semantic descriptions. The Card channel instead first retrieves heterogeneous Cards against the target statement, then follows their stored declaration links and retains only legal declarations. Formal and semantic rankings form the base pool, while the expanded pool additionally includes Card-linked declarations. Reciprocal-rank fusion combines the active channels, and the same frozen formal-only reranker selects at most ten ordered premises. Card prose is never an input to this selector.

3.3 What rich context changes

Rendering occurs only after premise selection. Compact context contains each selected declaration's name and formal Lean statement. Rich context preserves the same selected declarations and their order, but augments them with target-available semantic descriptions, structural relations, and Lean-verified repository-use evidence. Provenance retained during offline compilation is used to mask inaccessible or target-derived evidence before rendering.

This excerpt shows what changes for the same selected premise in the case study:

Compact	Rich (same premise)
`hconverge`: the formal `Filter.Tendsto` statement for the ADMM subsequence triple and a product neighborhood.	**Meaning:** convergence of (x_1, x_2, y) along the subsequence. **Concepts:** `Tendsto`, `atTop`, product neighborhoods. **Verified use:** `x1_subseq_converge` takes the first projection of `hconverge`.

The example visualizes the treatment; it is not evidence that any one added field causes success.

3.4 Frozen prover interface

Proof trajectories requested `gpt-5.6-sol`; every scientifically consumed response returned the same identifier (accessed September 18–19, 2026). The prover ran at temperature 0.2 under one fixed execution contract. It first proposes a blueprint—a high-level proof plan and possible

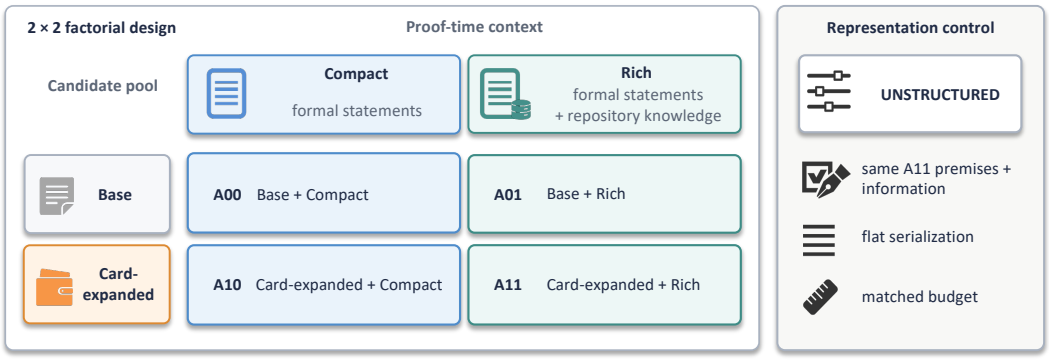

Figure 2: Rows vary candidate expansion; columns vary proof-time context. A00/A01 and A10/A11 are fixed-premise pairs. UNSTRUCTURED is a separate representation control, not a factorial cell.

decomposition—then generates Lean code for each intermediate node. A failed node can receive up to five bounded attempts; a trajectory uses at most three blueprint attempts and twelve nodes. Successful nodes are assembled, and Lean 4.24.0-rc1 accepts or rejects the final proof in the target's exact predeclaration environment.

4 Experimental Design

Benchmark and protocol. Optlib-50 contains 50 deterministically selected theorem targets from Optlib commit `d4f8f4e`, paired with Mathlib commit `c9e5385` and Lean 4.24.0-rc1. Selection occurred before model execution and did not use outcomes, difficulty, or retrieval quality. One *trajectory* is one prover run for one target under one condition; Pass@1 records whether its bounded run ends in a Lean-accepted proof. Completed unsuccessful runs and execution-unresolved runs are reported separately.

Conditions. Figure 2 crosses two choices. Candidate expansion (C) either uses formal-plus-semantic candidates or additionally includes declarations linked from retrieved Cards. Proof-time context (G) either shows compact formal statements or rich repository records after premise selection. This gives A00 (base candidates, compact), A01 (base candidates, rich), A10 (expanded candidates, compact), and A11 (expanded candidates, rich). Within each target, A00/A01 share the same ordered premises, as do A10/A11.

Unless otherwise noted, each active retrieval channel uses a cutoff of $K = 50$ before fusion, and the shared formal-only selector returns at most ten ordered premises. The tighter $K \in \{5, 10, 20, 50\}$ analysis reported later is a retrieval-only diagnostic reconstructed from frozen rankings and selector scores; it does not rerun the prover.

A fifth, UNSTRUCTURED condition uses A11's selected premises and the same target-available information, but flattens the rich hierarchy under a matched token budget. It is a representation control rather than a factorial cell.

Comparison roles. A00–A01 and A10–A11 compare compact and rich proof-time context while holding the ordered premise set fixed. A00–A10 probes candidate expansion under compact rendering, whereas A11–UNSTRUCTURED compares structured and flat presentation of the same selected premises and target-available information. The post-hoc R1 control separately compares compiled context with a relevance-preserving raw context under comparable context budgets. We analyze these comparisons separately because they intervene at different stages of the repository-to-prover interface.

Models and execution. All five conditions share the prover model, temperature, output limits, and bounded blueprint/node/repair contract specified in Section 3.4. Candidate selection uses the local `Qwen/Qwen3-Reranker-8B` checkpoint at revision `77d193c`; it scores only the target

and candidate formal interfaces. The offline semantic records were generated with requested model `gpt-5.4-mini` and required returned identity `gpt-5.4-mini-2026-03-17`. No condition fine-tunes the prover.

Analysis. Target-paired comparisons include scientifically complete pairs and use exact two-sided paired-binomial tests. We report full-denominator pass counts separately and give paired nonparametric-bootstrap percentile intervals for all prespecified risk differences; their finite empirical bootstrap distributions are enumerated exactly. The compiled-versus-raw R1 comparison is a post-hoc secondary control and is not substituted for the factorial.

5 RESULTS

The factorial study shows a larger observed difference along the proof-time-context axis than along the candidate-expansion axis, but neither factor is established as causally beneficial in this experiment. Full-denominator Pass@1 is 22/50 for A00, 27/50 for A01, 22/50 for A10, and 27/50 for A11 (Table 1); A10 and A11 additionally contain one and three execution-unresolved runs, respectively. Every prespecified paired contrast is statistically inconclusive, and the four-arm interaction is $+2.17$ points over 46 complete targets with a 95% paired-bootstrap interval of $[-21.74, 23.91]$ points. The repeated 22-versus-27 pattern is therefore a descriptive summary, not evidence that either intervention has a general effect. The clearest contrasts and the diagnostics that qualify them are analyzed separately below.

5.1 PROOF-TIME CONTEXT WITH PREMISES FIXED

Rich proof-time context has a positive but statistically inconclusive difference when premise identities and order are fixed. By design, A00 and A01 receive the same ordered premises for every target, while their proof-time packages differ. Compact A00 solves 22/50 targets and rich A01 solves 27/50. The paired outcomes are 17 both-pass, 10 A01-only, 5 A00-only, and 18 both-fail, giving a $+10$ percentage-point difference, a 95% paired-bootstrap interval of $[-4, 24]$ points, and exact two-sided $p = .301758$. Thus the observations motivate further study of proof-time repository context, but the interval includes zero and does not support a significance claim.

The fixed-premise contrast tests the complete rich-context package rather than any individual ingredient. Mean context length is 1,592 tokens in A00 and 15,431 in A01 over the same 50 targets. Rich rendering adds semantic descriptions, structural relations, and verified-use evidence, while also changing length and organization; the prompts are not token matched. Consequently, A01–A00 does not isolate whether content, length, formatting, or an interaction among them explains any trajectory difference. It establishes only that changing the full model-visible context package can change paired outcomes even when the selected premises themselves are held fixed.

The A00/A10 comparison is a descriptive same-selected-set calibration, not a stochastic noise-floor estimate. A00 and A10 share the same selected premise set on 48/50 targets and the same order on 42/50, yet their complete model-visible requests are byte-identical on 0/50 because condition and renderer metadata remain visible. Among the 47 scientifically evaluable same-set pairs, 9 are discordant: 5 pass only in A10 and 4 only in A00. This variation provides context for reading paired trajectories, but it cannot be subtracted from A01–A00 or treated as a strict rerun estimate. Appendix C gives the complete identity and outcome accounting, and Appendix B reports every prespecified contrast.

5.2 CANDIDATE-SIDE MANIPULATION AT $K = 50$

Card expansion only weakly changes the final premise interface at the original $K = 50$ cutoff. The legal earlier-same-file universe contains at most 50 declarations for 40/50 Optlib-50 targets and 83/100 Optlib-100 targets, so the per-channel budget often exhausts the retrieval task. At this cutoff, Card expansion changes the selected set for only 2/50 Optlib-50 targets (the ordered Top-10 list differs for 8/50), and just 1/53 genuinely Card-added candidate occurrences survives the shared formal-only Top-10 selector (Tables 3 and 4). Its candidate-side contrast should therefore be read as a weak-manipulation condition at $K = 50$, rather than as a general test of whether Card expansion can alter premise selection.

Table 1: Optlib-50 outcomes. Mean context tokens are computed over the same 50 frozen contexts with the `o200k_base` tokenizer. UNSTRUCTURED is a separate representation control.

Arm	Expand	Context	Mean tokens	PASS	Fail	Unresolved	Pass@1
A00	No	Compact	1,592	22	28	0	44%
A01	No	Rich	15,431	27	23	0	54%
A10	Yes	Compact	1,587	22	27	1	44%
A11	Yes	Rich	17,277	27	20	3	54%
UNSTRUCTURED	Yes	Flat	17,398	27	23	0	54%

Table 2: Observable source-proof occurrences recovered before the final selector. FS uses formal+semantic retrieval; CFS also uses Card expansion.

Cohort	Condition	$K = 5$	$K = 10$	$K = 20$	$K = 50$
Optlib-50	FS	24/37	28/37	33/37	37/37
Optlib-50	CFS	25/37	30/37	35/37	37/37
Optlib-100	FS	40/50	45/50	47/50	49/50
Optlib-100	CFS	41/50	46/50	48/50	49/50

For the observable source-proof labels, measured coverage loss occurs after candidate discovery and before the final Top-10 interface. Before final selection, both formal-plus-semantic (FS) retrieval and its Card-expanded counterpart (CFS) recover 37/37 observable source-proof occurrences in Optlib-50 (Table 2). Their final Top-10 selections each retain 30/37. This stagewise accounting localizes where the measured labels disappear, but it does not assign causal blame to the selector or show that the omitted occurrences are necessary for proving. Source-proof references are observable and non-exhaustive relevance labels for the restricted same-file task; other premises may be useful, and imported declarations remain available to Lean outside this retrieval evaluation.

5.3 Retrieval-budget diagnostic

Retrieval budget strongly controls how much Card expansion can change the final premise interface. For Optlib-50, FS and CFS produce different final selected sets on 32/50 targets at $K = 5$, 18/50 at $K = 10$, 10/50 at $K = 20$, and only 2/50 at $K = 50$ (Table 3). The same pattern is even more visible in the survival of genuinely Card-added candidates through the shared formal-only Top-10 selector: 69/74 (93.24%) survive at $K = 5$, compared with 34/100 (34.00%) at $K = 10$, 15/111 (13.51%) at $K = 20$, and only 1/53 (1.89%) at $K = 50$ (Table 4). Thus, Card retrieval can materially change which declarations reach the final premise interface under tight candidate budgets, whereas the generous $K = 50$ cutoff largely suppresses that difference before proof generation begins.

The same budget-dependent pattern appears on the larger Optlib-100 retrieval cohort. Card expansion changes the final selected set for 64/100 targets at $K = 5$, 34/100 at $K = 10$, 10/100 at $K = 20$, and 6/100 at $K = 50$. Card-added candidate survival likewise falls from 131/140 (93.57%) at $K = 5$ to 49/161 (30.43%) at $K = 10$, 15/173 (8.67%) at $K = 20$, and 10/90 (11.11%) at $K = 50$. This second cohort is retrieval-only rather than an end-to-end proving evaluation, but it shows that the dependence of expansion leverage on candidate budget is not specific to the 50 targets used in the factorial experiment.

Stronger candidate-set perturbation produces only modest gains on the observable source-proof labels. For Optlib-50, CFS improves pre-selector source-proof recovery over FS from 24/37 to 25/37 at $K = 5$, from 28/37 to 30/37 at $K = 10$, and from 33/37 to 35/37 at $K = 20$; at $K = 50$, both recover 37/37 (Table 2). Optlib-100 shows the same qualitative pattern: CFS adds one observable occurrence over FS at $K = 5$, $K = 10$, and $K = 20$, while both recover 49/50 at $K = 50$. Taken together, the diagnostics separate two effects that are easy to conflate: tighter budgets make Card expansion much more capable of changing the candidate interface, while the additional coverage measured by source-proof references remains small. These results characterize

Table 3: Targets whose final selected sets differ with Card expansion in the precommitted budget analysis.

Cohort	$K = 5$	$K = 10$	$K = 20$	$K = 50$
Optlib-50	32/50	18/50	10/50	2/50
Optlib-100	64/100	34/100	10/100	6/100

Table 4: Selected-set differences and survival of genuinely Card-added candidates through the shared formal-only Top-10 selector.

Cohort	K	Different sets	Card-added retained/added	Survival
Optlib-50	5	32/50	69/74	93.24%
Optlib-50	10	18/50	34/100	34.00%
Optlib-50	20	10/50	15/111	13.51%
Optlib-50	50	2/50	1/53	1.89%
Optlib-100	5	64/100	131/140	93.57%
Optlib-100	10	34/100	49/161	30.43%
Optlib-100	20	10/100	15/173	8.67%
Optlib-100	50	6/100	10/90	11.11%

retrieval-side behavior; the end-to-end proving effect of the stronger tight-budget intervention is not evaluated here.

Appendix D gives the complete legal-universe and scope accounting for both cohorts.

5.4 Representation and compilation controls

Structured rich context shows no aggregate advantage over the tested flat representation control. A11 and UNSTRUCTURED use the same A11 premises and target-available information under the matched token budget defined in Section 4; they differ in structured versus flat serialization. Both record 27/50 full-denominator passes. Across 47 scientifically complete pairs, A11 minus UNSTRUCTURED is $+2.13$ points with a 95% paired-bootstrap interval of $[-10.64, 14.89]$ and exact $p = 1$. This control does not support a universal advantage for structured presentation, nor does it show that the two representations are equivalent.

Compiled context likewise shows no aggregate advantage over the budget-comparable raw-context control. The post-hoc R1 comparison holds targets and A01 premises fixed and compares newly rendered compiled context with a relevance-preserving raw baseline. Compiled and raw solve 24/50 and 26/50 targets; the paired outcomes are 15 both-pass, 9 compiled-only, 11 raw-only, and 15 both-fail, yielding compiled minus raw of -4 points and exact $p = .824$. The contexts are budget-comparable rather than exactly length matched: the compiled/raw token-ratio mean and median are 0.965 and 1.000, with 46/50 ratios at least 0.90. R1 therefore establishes neither raw superiority nor compiled/raw equivalence; it only rules out an observed aggregate advantage for compilation in this tested control.

5.5 Case Study: One Proof in Detail

One mechanically selected fixed-premise case illustrates how rich repository information can align with a successful proof trajectory without establishing why it succeeded. The target asks whether the second ADMM iterate's error converges to zero along a subsequence. Compact and rich conditions receive the same ten premises in the same order, but their outcomes differ: the compact trajectory exhausts five node attempts, whereas the rich trajectory succeeds on its fourth attempt. Their contexts contain 4,433 and 30,438 tokens, respectively, so this case reflects the full rich package rather than a length-matched intervention. Figure 3 summarizes the comparison.

The accepted rich proof uses a projection pattern that is explicit in its added repository knowledge. The selected lemma `hconverge` states that the subsequence of the three ADMM com-

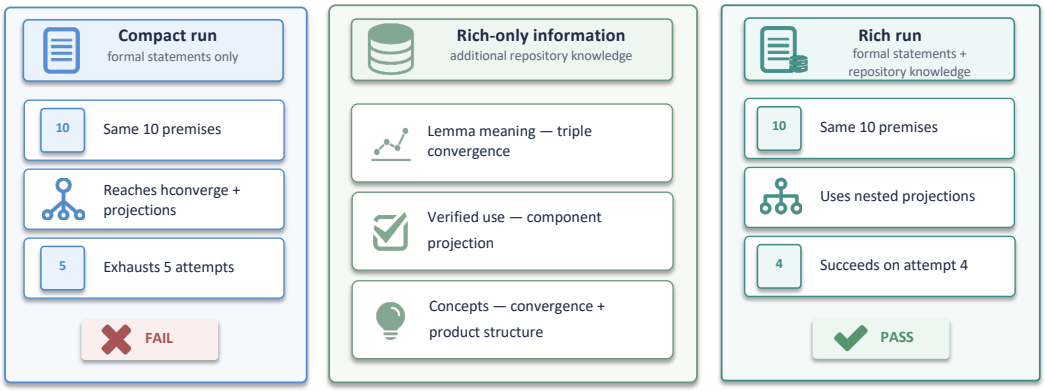

Figure 3: One fixed-premise pair. The accepted rich proof uses a projection pattern visible in rich context; the compact run reaches the same broad idea. This establishes neither necessity nor causal mediation.

ponents (x_1, x_2, y) converges to a triple of limits. The successful trajectory takes nested product projections from this triple convergence, subtracts a constant with `Tendsto.sub_const`, and simplifies; its accepted proof syntactically uses `hconverge`. Rich context describes the triple-convergence meaning, relevant `Tendsto` and product-neighborhood concepts, and a target-available verified example that extracts a component by projection. This establishes that the successful trajectory used a selected premise in a way aligned with information visible in rich context.

Across discordant fixed-premise pairs, outcome changes are more often associated with different proof composition than with a directly traceable use of one added field. The compact run in the positive case also reaches `hconverge` and projection ideas before exhausting its attempts, so the comparison does not establish that any added field was necessary or mediated success. The mechanically selected opposite-direction case receives the same premises but passes with compact context and fails with rich context: the compact run finds a direct `inner_gradient` composition, while the rich run follows a different unsuccessful composition. Across all ten A01-only passes, the frozen trajectory audit identifies one case with traceable node-level alignment to added repository information, seven shared-premise/different-composition cases, and two cases without traceable alignment; all five A00-only passes fall into the shared-premise/different-composition category. Thus, the detailed case is part of a broader pattern in which changing proof-time repository context can coincide with different ways of composing the same selected premises, rather than simply enabling one previously unavailable declaration. The complete selection rule and discordant-case categories appear in Appendix H.

6 Conclusion

We introduced Library Card, an automated repository knowledge interface for Lean theorem proving that organizes repository-specific information and exposes it at different stages of the proving pipeline. By separating candidate availability from post-selection context, Library Card enables repository knowledge to be analyzed across three stages: candidate discovery, premise selection, and proof-time use. On Optlib-50, the baseline retriever recovers all observable source-proof premises at the evaluated cutoff before final selection, while the subsequent selector retains only a subset of them. When the selected premises and their order are held fixed, richer proof-time context changes the observed solve rate from 44% to 54%. Together, these results show that repository-aware theorem proving should not be treated as a single retrieval problem: useful knowledge must first be retrieved, then retained through selection, and finally presented in a form that supports proof generation. Library Card provides a controlled interface for studying these stages independently and offers a practical foundation for designing more effective connections between formal repositories and frozen theorem provers.

AI USE STATEMENT

The evaluated system used named models to generate semantic records, reranker scores, and proof proposals; deterministic heuristics produced Cards. Lean was the sole authority for proof acceptance. Generative AI also assisted research design, implementation, analysis, literature verification, and writing. The authors checked the resulting claims against the pinned artifacts and primary sources and take responsibility for the paper.

REPRODUCIBILITY STATEMENT

The evaluation pins Optlib commit `d4f8f4e`, Mathlib commit `c9e5385`, Lean 4.24.0-rc1, requested prover identifier `gpt-5.6-sol` (also returned by every scientifically consumed response; accessed September 18–19, 2026), `Qwen3-Reranker-8B` revision `77d193c`, and semantic compiler `gpt-5.4-mini-2026-03-17`. Reproducing the retrieval analysis requires the pinned repository snapshot, stored Cards and semantic records, channel rankings, and selector scores. Reproducing proof outcomes additionally requires the frozen prompts, prover contract, execution logs, and Lean environments. An anonymized code release is available at `https://anonymous.4open.science/r/Library-Card` and is also provided in the supplementary material.

REFERENCES

Justin Asher. Leanexplore: A search engine for lean 4 declarations. *arXiv preprint arXiv:2506.11085*, 2025.

Lasse Blaauwbroek, Mirek Olšák, Jason Rute, Fidel Ivan Schaposnik Massolo, Jelle Piepenbrock, and Vasily Pestun. Graph2Tac: Online Representation Learning of Formal Math Concepts. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 4046–4076. PMLR, 2024. URL https://proceedings.mlr.press/v235/blaauwbroek24a.html.

Jui-Hui Chung, Ziyang Cai, Zihao Li, Qishuo Yin, Rohit Agarwal, Simon Park, Rodrigo Porto, Narutatsu Ri, Ziran Yang, Shange Tang, Xingyu Dang, Hongzhou Lin, Mengdi Wang, Danqi Chen, Chi Jin, Liam H Fowl, and Sanjeev Arora. Goedel-Architect: Streamlining formal theorem proving with blueprint generation and refinement, 2026. URL https://arxiv.org/abs/2606.06468.

Guoxiong Gao, Haocheng Ju, Jiedong Jiang, Zihan Qin, and Bin Dong. A semantic search engine for mathlib4. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8001–8013, 2024.

Guoxiong Gao, Zeming Sun, Jiedong Jiang, Yutong Wang, Jingda Xu, Peihao Wu, Bryan Dai, and Bin Dong. LeanSearch v2: Global premise retrieval for Lean 4 theorem proving, 2026. URL https://arxiv.org/abs/2605.13137.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Jiewen Hu, Thomas Zhu, and Sean Welleck. minictx: Neural theorem proving with (long-) contexts. In *International Conference on Learning Representations*, volume 2025, pp. 10106–10130, 2025.

Adarsh Kumarappan, Mohit Tiwari, Peiyang Song, Robert Joseph George, Chaowei Xiao, and Anima Anandkumar. LeanAgent: Lifelong learning for formal theorem proving. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Uo4EHT4ZZ8.

Chenyi Li, Ziyu Wang, Wanyi He, Yuxuan Wu, Shengyang Xu, and Zaiwen Wen. Formalization of complexity analysis of the first-order algorithms for convex optimization, 2024. URL https://arxiv.org/abs/2403.11437.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, et al. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In *International Conference on Learning Representations*, volume 2026, pp. 11793–11818, 2026.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Jialin Lu, Kye Emond, Kaiyu Yang, Swarat Chaudhuri, Weiran Sun, and Wuyang Chen. Lean finder: Semantic search for mathlib that understands user intents. In *International Conference on Learning Representations*, volume 2026, pp. 155005–155040, 2026.

Maciej Mikuła, Szymon Tworkowski, Szymon Antoniak, Bartosz Piotrowski, Qiaochu Jiang, Jin Zhou, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. Magnushammer: A transformer-based approach to premise selection. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/ab5f5f22e3e09f4424592ffb06840ab0-Abstract-Conference.html.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, pp. 625–635. Springer, 2021.

Job Petrovčič, David Eliecer Narvaez Denis, and Ljupčo Todorovski. Combining textual and structural information for premise selection in Lean, 2025. URL https://arxiv.org/abs/2510.23637.

Santhana Srinivasan R and Maithilee Patawar. LAMP: Lean-based agentic framework with MCP and proof repair, 2026. URL https://arxiv.org/abs/2606.28841.

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.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean Copilot: Large language models as copilots for theorem proving in Lean. In *Proceedings of the International Conference on Neuro-symbolic Systems*, volume 288 of *Proceedings of Machine Learning Research*, pp. 144–169. PMLR, 2025. URL https://proceedings.mlr.press/v288/song25a.html.

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. An in-context learning agent for formal theorem-proving. *arXiv preprint arXiv:2310.04353*, 2023.

Kyle Thompson, Nuno Saavedra, Pedro Carrott, Kevin Fisher, Alex Sanchez-Stern, Yuriy Brun, João F Ferreira, Sorin Lerner, and Emily First. Rango: Adaptive retrieval-augmented proving for automated software verification. *arXiv preprint arXiv:2412.14063*, 2024.

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively building formal proofs with informal reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 38998–39031, 2026.

Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. *Advances in neural information processing systems*, 30, 2017.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *International Conference on Learning Representations*, volume 2025, pp. 72274–72303, 2025.

Yutong Xin, Qiaochu Chen, Greg Durrett, and Işıl Dillig. VeriSoftBench: Repository-scale formal verification benchmarks for Lean. In *Conference on Language Modeling*, 2026. URL https://arxiv.org/abs/2602.18307.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612, 2023.

Meiru Zhang, Philipp Borchert, Milan Gritta, and Gerasimos Lampouras. Drift: decompose, retrieve, illustrate, then formalize theorems. In *International Conference on Learning Representations*, volume 2026, pp. 6828–6857, 2026.

Xiyuan Zhou, Zhuoqi Li, Xinlei Wang, Yirui He, Yuhao Wu, Yuheng Cheng, Yan Xu, Junhua Zhao, and Jinjin Gu. Repro: Proof-verified benchmark rewriting for reliable evaluation of llm mathematical problem solving. *arXiv preprint arXiv:2609.00062*, 2026.

Thomas Zhu, Joshua Clune, Jeremy Avigad, Albert Qiaochu Jiang, and Sean Welleck. Premise Selection for a Lean Hammer. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2506.07477.

APPENDIX

A METHOD DETAILS

The frozen corpus contains 1,518 Cards: 1,301 declaration, 181 concept, 22 theorem-family, and 14 strategy Cards. Declaration Cards organize formal and repository-derived information around individual Lean declarations; concept and theorem-family Cards group related declarations, while strategy Cards capture recurring proof-use patterns. All Card types ultimately expose links to concrete repository declarations for candidate expansion.

The repository graph and semantic compiler are parallel stored products: semantic records do not generate Cards. Repository-use relations are extracted from existing repository proofs and retained with source provenance for verified-use evidence. The frozen build and target-time projections use this provenance to mask evidence unavailable before the target.

B FULL OPTLIB-50 FACTORIAL STATISTICS

Table 5: Prespecified paired contrasts over scientifically complete pairs. Differences are treatment minus control; intervals are 95% paired nonparametric-bootstrap percentile intervals, evaluated by exact finite enumeration. All exact tests are statistically inconclusive.

Contrast	Complete N	Paired diff. (pp)	95% CI (pp)	Exact p	Interpretation
A11–A00	47	10.64	$[-2.13, 23.40]$	0.179688	Inconclusive
A10–A00	49	2.04	$[-10.20, 14.29]$	1	Inconclusive
A01–A00	50	10.00	$[-4.00, 24.00]$	0.301758	Inconclusive
A11–A10	46	8.70	$[-6.52, 23.91]$	0.387695	Inconclusive
A11–A01	47	4.26	$[-10.64, 19.15]$	0.774414	Inconclusive
A11–UNSTRUCTURED	47	2.13	$[-10.64, 14.89]$	1	Inconclusive

The complete arm outcomes are given in Table 1. For the fixed-premise A01–A00 comparison, 17 pairs pass in both conditions, 10 pass only with rich context, 5 only with compact context, and 18 fail in both. The paired risk difference is +10 points; exact evaluation of the finite paired- bootstrap distribution gives a 95% percentile interval of $[-4, 24]$ points. The four-arm interaction is +2.17 points over 46 complete targets, with a 95% bootstrap interval of $[-21.74, 23.91]$ points. Execution-unresolved outcomes remain distinct from proof failures.

C A00/A10 INPUT-IDENTITY CALIBRATION

A00 and A10 have the same selected premise set on 48/50 targets and the same ordered list on 42/50. Their complete model-visible requests are byte-identical on 0/50: at minimum, the prompt exposes the A00 versus A10 pilot-mode label, and most nonempty contexts also expose different renderer-method labels.

Across all 50 targets, paired outcomes are 17 both-pass, 5 A10-only, 4 A00-only, 23 both-fail, and one unresolved. Restricting to the 48 same-set targets gives 17 both-pass, 5 A10-only, 4 A00-only, 21 both-fail, and one unresolved; 9/47 scientifically evaluable pairs are discordant. Because the model-visible inputs are not identical, this is a same-selected-set descriptive calibration rather than a strict stochastic rerun estimate. No noise subtraction or adjusted treatment estimate is made.

D RETRIEVAL AUDIT AND BUDGET ANALYSIS

All retrieval metrics use source-proof premise recall: references visible in recoverable source proofs, not exhaustive annotations of every premise that could prove a theorem. Zero-denominator and unresolved targets receive no artificial recall value.

The analysis fixed $K \in \{5, 10, 20, 50\}$ before aggregate computation and reconstructed pools and selections from stored channel rankings and formal-only reranker scores. It made no provider, re-

trieval, reranker, prover, or Lean call and did not tune cutoffs after observing results. Main-text Tables 2 and 4 report the complete budget grids for discovery coverage and Card-added retention.

Table 6: Earlier same-file candidate-set distribution.

Cohort	Min	Q1	Median	Mean	Q3	Max
Optlib-50	0	5.75	16.5	26.26	37	91
Optlib-100	0	5	14	25.43	39.75	118

At $K = 50$, FS/CFS recover 37/37 versus 37/37 observable occurrences in Optlib-50 and 49/50 versus 49/50 in Optlib-100 before selection. The full Top-10 selections recover 30/37 in both Optlib-50 conditions and 45/50 in both Optlib-100 conditions.

Among exactly mapped repository-reference occurrences, 45/45 in Optlib-50 and 88/88 in Optlib-100 are earlier declarations in the target's source file. Most other used-constant occurrences (4,855/4,925 and 13,011/13,170) are classified upstream as Mathlib/imported external and lie outside this retrieval task; the extractor does not support a unique declaration count across all categories.

E Post-hoc R1 Raw-Context Control

R1 compares newly rendered COMPILED_BUDGET_MATCHED (24/50) with RAW_RELEVANT (26/50) on the same targets and A01 premises. The paired table is 15 both-pass, 9 compiled-only, 11 raw-only, and 15 both-fail; compiled minus raw is -4 points with exact two-sided $p = .824$. Mean and median compiled/raw token ratios are 0.965 and 1.000, and 46/50 are at least 0.90. This post-hoc comparison establishes neither equivalence nor superiority.

F Historical Evaluation

A separate earlier setup evaluated two disjoint cohorts totaling 190 targets. Library Card solved 82/190 (43.16%) and its matched baseline 71/190 (37.37%), an observed difference of +5.79 points. This evidence is not pooled with Optlib-50.

Among 170 scientifically complete pairs, 53 pass in both arms, 73 fail in both, 27 pass only with Library Card, and 17 only with the baseline. The paired difference is +5.88 points ($p = .174$); 20 pairs are incomplete. The associated audit is observational and does not establish mediation.

G Historical Mechanism Audit

The historical audit classifies 44 discordant complete pairs using an exclusive frozen hierarchy. For H1-only passes, category counts are $2/15/0/0/0/0/10$: 2 accepted proofs use a Card-added declaration absent from H0, 15 use shared premises while richer guidance is available, and 10 retain insufficient or unclear provenance. For H0-only passes the counts are $0/0/0/1/0/7/9$: one proof-composition difference, seven cases without a Card-specific trace, and nine with insufficient provenance. These observations are separate from Optlib-50 and establish neither necessity, harm, nor mediation.

H Context-Trajectory Audit

All 50 A00/A01 pairs passed treatment-integrity checks. Positive and opposite-direction cases were selected before narrative inspection by an objective provenance-completeness score and canonical-name tie break. In the positive case, identical ordered premises yield 4,433 versus 30,438 context tokens and FAIL/PASS. Both blueprints have one node; the rich run succeeds on its fourth attempt after nested projections and `Tendsto.sub_const`, while the compact run fails after five attempts. The accepted proof syntactically uses `hconverge`; this is utilization and trajectory alignment, not necessity or mediation.

The opposite case has identical premises, 3,223 versus 17,719 context tokens, and PASS/FAIL. Across the 10 rich-only cases, the mutually exclusive audit finds one traceable node-level alignment, seven shared-premise/different- composition traces, and two with no traceable alignment. All five compact-only cases have shared-premise/different-composition traces.

I Case-Study Provenance

The positive target is `ADMM_Converge_Proof.e2_subseq_converge_zero`; the opposite target is `ADMM_Converge_Proof.ADMM_iter_process1_eq3_2_1`. Their full context differences, blueprint comparisons, node chronologies, and independent validation are retained with the evaluation artifacts. They support qualitative illustration only.

J Reproducibility

Pinned software and models. The evaluation uses Optlib commit `d4f8f4ee9a4516b74c1b5016b6aa1f142d9d054a`, Mathlib commit `c9e5385874ef9855f0b43408ad9867e9c74853d3`, and Lean `v4.24.0-rc1`. Proof trajectories requested `gpt-5.6-sol`, and every scientifically consumed response returned that identifier (accessed September 18–19, 2026), at temperature 0.2 with blueprint/node output limits of 8,192/4,096 tokens, at most three blueprint attempts, five attempts per node, and twelve nodes. Premise selection uses the local `Qwen/Qwen3-Reranker-8B` checkpoint at revision `77d193c791ed757ca307ee72715aa132723da912`. Semantic records require returned model identity `gpt-5.4-mini-2026-03-17`.

Anonymous code release. An anonymized code release is available at `https://anonymous.4open.science/r/Library-Card` and is also provided in the supplementary material. It includes deterministic Card construction, same-file-earlier legality construction, formal/semantic/Card retrieval interfaces, Card-to-declaration expansion, reciprocal-rank fusion, frozen-score premise selection, compact/rich/flat context rendering, and code for validating and reconstructing the released aggregate analyses. The accompanying documentation maps paper components to their implementations and identifies components that are executable, interface-only, artifact-reconstruction-only, or not released.

Scope of reproduction. Provider-free Library Card construction, retrieval-side analyses, context rendering, and supported statistical reconstructions can be executed from the released materials. Full model-dependent reproduction additionally requires semantic-generation and reranker inference, exact prompts and materialized contexts, the blueprint/node/repair execution engine, model access, and pinned Lean environments. Target-time Rich-evidence masking, Lean-verified-use materialization, private provider infrastructure, and private row-level execution artifacts are not included in the anonymous release.