Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

EXPLAINABLE AI FOR MATHEMATICS: PROOFS AS CODE WITH KNOWLEDGE GRAPH AND DOMAIN ONTOLOGY SUPPORT

A. P. Khalov[1,2]*, O. M. Ataeva[1] and N. P. Tuchkova[1]

[1] *Federal Research Center "Computer Science and Control", Russian Academy of Sciences, Moscow, 119333 Russia*
[2] *Moscow Institute of Physics and Technology (National Research University), Dolgoprudny, Moscow Region, 141701 Russia*

khalov.a@phystech.edu, oataeva@frccsc.ru, ntuchkova@frccsc.ru

ABSTRACT

We investigate whether structured knowledge retrieval from a mathematical library's dependency graph can improve neural theorem proving at inference time while maintaining explainability of the retrieved context. Through a controlled ablation study on the miniF2F-Test benchmark (197 tasks, 8 non-chain-of-thought modes, $K{=}8$ attempts per mode), we find that graph-based retrieval augmentation significantly improves proof generation on hard problems – those where the base model's parametric knowledge is insufficient – while having no measurable effect on problems the model already solves reliably. On 109 hard tasks, the best graph-augmented mode nearly triples the baseline success rate (pass@1: $3.56\% \to 10.42\%$, $+6.8$ percentage points, $p{=}0.001$, Wilcoxon signed-rank test). Deterministic pattern-based graph entry points outperform LLM-generated ones ($11\times$ faster, higher accuracy). The retrieved graph context is fully traceable: each hint maps to a specific edge in the Mathlib dependency graph, enabling the user to verify *why* particular lemmas were suggested. The approach is training-free, composable with any base prover, and adds negligible computational overhead.

Keywords: explainable AI, neural theorem proving, knowledge graph, domain ontology, mathlib dependency graph, Lean 4, graph RAG

1 INTRODUCTION

The formalization of mathematical knowledge in interactive theorem provers such as Lean 4 (de Moura and Ullrich, 2021) has created a unique opportunity for AI-assisted mathematics. Under the Curry–Howard correspondence (Howard, 1980; Wadler, 2015), proofs are programs and propositions are types: verifying a mathematical proof reduces to type-checking a program. This equivalence enables rigorous, machine-checkable verification of mathematical reasoning – a property absent from natural-language proofs. In Martin-Löf's intuitionistic type theory (Martin-Löf, 1984), the correspondence extends to dependent types, enabling propositions that quantify over values, and Lean 4 implements precisely this framework. Its mathematical library, Mathlib (The mathlib Community, 2020), contains over 213,000 formally verified declarations, organized as a typed dependency graph where each declaration's type signature and proof body reference other declarations.

Recent neural theorem provers (Yin et al., 2025; Wang et al., 2025; ByteDance Seed AI4Math, 2025) have achieved remarkable pass rates on standard benchmarks by training large language models to generate formal proofs. However, these systems operate as black boxes: when a prover fails, the user receives no insight into *why* the proof attempt was unsuccessful, and when it succeeds, the connection between the generated proof and the underlying mathematical knowledge remains

*Corresponding author.

Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

opaque. This opacity undermines the very premise of formal mathematics – that proofs should be not merely correct, but *understandable*.

We address this problem through *explainable context delivery*: augmenting the prover's prompt with structured hints retrieved from a knowledge graph (KG) built from Mathlib's dependency structure. Each hint is traceable to a specific graph edge – a `usesInType` or `usesInValue` relation between Mathlib declarations – enabling the user to inspect and verify the retrieved context. This approach operates in the *pass@1–3 regime* (Yang et al., 2025): practical settings where computational budgets limit the number of proof attempts, and each attempt should be informed by the best available context. Unlike tree-search provers that require hundreds or thousands of proof attempts, our method provides targeted, interpretable assistance in low-budget scenarios.

Our contributions are as follows. First, we present a **controlled ablation study** comparing 8 non-chain-of-thought modes on miniF2F-Test (Zheng et al., 2022) (197 tasks, $K{=}8$), with paired statistical tests (Wilcoxon signed-rank) and confidence intervals – more rigorous than typical proof generation evaluations. Second, we demonstrate a **strongly asymmetric effect**: graph-augmented context nearly triples the baseline success rate on hard tasks ($+6.8$pp, $p{<}0.001$) while showing no effect on easy tasks, with the hard task set naturally capturing 84–91% of competition-level problems (IMO, AIME). Third, we show that **deterministic pattern-based** graph entry points outperform LLM-generated seeds ($+5.3$pp vs $+2.5$pp, $11\times$ faster, zero LLM calls), demonstrating that simple interpretable rules beat neural generation for knowledge graph navigation. Fourth, our approach is **training-free and composable**, requiring no fine-tuning and applicable on top of any base prover as an inference-time enhancement.

The paper is organized as follows. Section 2 reviews proofs-as-code, neural theorem proving, and retrieval-augmented approaches. Section 3 describes the knowledge graph construction and multimodal representation. Section 4 details the eight context delivery modes. Sections 5–6 present the experimental setup and quantitative results. Section 7 provides a detailed case study demonstrating the explainability mechanism. We discuss implications in Section 8, limitations in Section 9, and conclude in Section 10.

2 Background and Related Work

2.1 Proofs as Code

The Curry–Howard correspondence (Howard, 1980) establishes a deep structural analogy between formal proofs and typed programs: propositions correspond to types, and proofs to terms inhabiting those types. Wadler (Wadler, 2015) provides a modern exposition of this correspondence and its implications for programming language theory and formal verification. Lean 4 (de Moura and Ullrich, 2021) is a dependently typed programming language and interactive theorem prover that implements this correspondence. Its mathematical library, Mathlib (The mathlib Community, 2020), has grown into the largest unified collection of formally verified mathematics, covering algebra, analysis, topology, number theory, and combinatorics. This "proofs as code" perspective has a practical consequence: a proof assistant can *verify* any generated proof by type-checking, regardless of the proof's origin. The verification is sound, complete for the underlying logic, and independent of the generation method, making neural theorem proving uniquely amenable to rigorous evaluation – unlike natural-language mathematics, there is no ambiguity about correctness.

2.2 Neural Theorem Proving

Modern neural theorem provers fall into two broad categories. *Tree-search provers* such as Aristotle (The Harmonic Team, 2025) and Seed-Prover (ByteDance Seed AI4Math, 2025) decompose proofs into tactic steps and explore the proof tree via Monte Carlo tree search or best-first search. *Whole-proof provers* such as DeepSeek-Prover-V2 (Yin et al., 2025) and Kimina-Prover (Wang et al., 2025) generate complete proofs in a single pass, relying on the model's parametric knowledge to produce a valid tactic sequence. Recent advances have been rapid: Goedel-Prover (Lin et al., 2025) achieves 57.6% on miniF2F-Test at pass@32 through synthetic data augmentation and supervised fine-tuning, while DeepSeek-Prover-V2-7B (Yin et al., 2025), building on DeepSeekMath (Shao et al., 2024), reaches 55.5% at pass@1 through reinforcement learning for subgoal decomposition. At higher sampling budgets, Leanabell-Prover-V2 (Wu et al., 2025) achieves 78.2% at pass@128,

Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

and Hilbert (Varambally et al., 2025) demonstrates that informal reasoning can guide formal proof construction. Our work focuses on the whole-proof setting, which is more interpretable: the model's output is a single, complete proof that can be read and understood as a coherent argument.

2.3 Retrieval-Augmented Proof Generation

Retrieval-augmented generation (RAG) (Lewis et al., 2020) has proven effective across many knowledge-intensive tasks, and several recent works apply it to theorem proving. REAL-Prover (Shen et al., 2025) fine-tunes a general-purpose model on retrieval-augmented training data, achieving 54.1% on miniF2F-Test at pass@64. LeanDojo (Hsiang et al., 2025) provides infrastructure for premise retrieval in Lean 4 environments. Herald (Gao et al., 2025) constructs a natural-language-annotated Lean 4 dataset for training retrieval models. LeanRAG (Zhang et al., 2025) proposes hierarchical retrieval over knowledge graphs for proof generation. Semantic search engines for Mathlib (Gao et al., 2024; Asher, 2025) enable similarity-based retrieval of relevant declarations, while ATLAS (Liu et al., 2025b) addresses autoformalization as a complementary direction. Our approach differs in three respects: we augment an *already specialized* prover at inference time, requiring no additional training; we retrieve from a *typed dependency graph* rather than a flat document store, providing structured context classified by usage pattern; and we evaluate with *paired statistical tests* on a controlled ablation, isolating the contribution of each retrieval component.

2.4 Knowledge Graphs for Mathematics

Structured representations of mathematical knowledge have a long history, from standards such as MathML (World Wide Web Consortium, 2010) and OpenMath (Buswell et al., 2004) to modern ontological systems. OntoMathPRO (Nevzorova et al., 2014) provides a linked data ontology for mathematical concepts. Serebryakov and Ataeva (Serebryakov and Ataeva, 2021) develop an ontology-based approach to modeling the mathematical domain in digital libraries. The SPAR ontologies (Peroni and Shotton, 2018) and the Open Research Knowledge Graph (Brack et al., 2020) offer frameworks for scholarly knowledge representation more broadly, while formal benchmarks (Zheng et al., 2022; Liu et al., 2025c;a) provide standardized evaluation of mathematical reasoning. AI-assisted scientific research more generally (Chen et al., 2025) has seen rapid progress, though the specific challenge of formal mathematics requires specialized approaches. Our work extends this line by constructing a knowledge graph directly from Mathlib's dependency structure, where nodes are formal declarations and edges represent typed usage relations, capturing the operational structure of mathematical knowledge – which lemmas are used to prove which theorems, and in what capacity.

3 Architecture

3.1 SciLib Ontology and Three-Level Model

The infrastructure is built on the SciLib ontology – a modular meta-level model designed to describe scientific knowledge objects and the processes of their storage, retrieval, and use in computational pipelines. Although in this work only the mathematics domain is considered, the ontology was designed to be domain-general: new subject domains, representations, or computational components can be added without changing the basic semantic invariants. The ontology implements a three-level separation of knowledge structure. The *interpretation level* describes the abstract meaning of a scientific object independently of its form of expression – for instance, the concept "the fundamental theorem of calculus" as understood by a mathematician. The *representation level* fixes the concrete form of expression of this meaning: formal code in Lean 4, natural language text, symbolic notation, or visual rendering. The *resource level* corresponds to the material or computable carrier of representation, including files, database entries, and executable artifacts. Each mathematical statement exists simultaneously at all three levels, linked by a URI that serves as the canonical identifier across modalities. This separation realizes the principle of invariance and enables scalability to an arbitrary number of domains, support for alternative interpretations, and formal analysis of scientific knowledge without mixing levels of abstraction. The three-level model is detailed in a companion submission; here we focus on its role as the foundation for our knowledge graph and multimodal retrieval.

Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

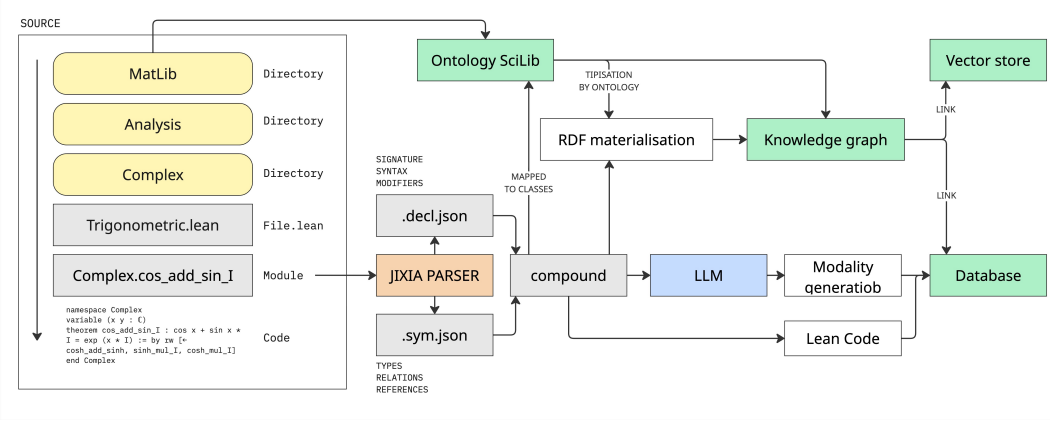

Figure 1: Knowledge graph construction pipeline. Mathlib source is compiled by Lean 4, producing intermediate declaration files. The JIXIA parser extracts declarations, metadata, and structural dependencies, which are mapped to the SciLib ontology and materialized as 66M RDF triples in GraphDB.

3.2 Knowledge Graph Materialization

The knowledge graph is constructed by materializing the Lean 4 Mathlib library into the SciLib ontological model through a reproducible multi-stage pipeline (Figure 1). The Mathlib source is compiled by Lean 4, producing intermediate representations (`.decl.json` and `.sym.json` files). A dedicated parser (JIXIA) extracts three data types from these files: string metadata (names, docstrings), Lean formal code (type signatures, proof bodies), and structural dependencies between declarations. Additionally, the hierarchical organization of the library is analyzed, reflecting the thematic division of mathematical knowledge into 660 subclasses of the Domain class, each automatically annotated using an LLM.

All extracted entities and connections are then mapped to the SciLib ontological model and materialized as an RDF graph. The key methodological assumption is that formal provability in Mathlib serves as a truth criterion: if a statement φ is provable in the formal system, then its interpretation in the mathematical domain is considered true:

$$M_{\text{MathLib}} \vdash \varphi \;\Rightarrow\; I_{\text{Math}}(\varphi) = \text{True} \tag{1}$$

From the structural dependencies, we materialize several types of directed edges. The two most important for retrieval are $\texttt{usesInType}(A, B)$, indicating that declaration A appears in the type signature of B (i.e., A is part of the *statement* of B), and $\texttt{usesInValue}(A, B)$, indicating that A is referenced in the proof body of B (i.e., A is used as a *tactic or lemma* in proving B). Additional relations include `studiedIn` (linking statements to mathematical domains, 180K relations) and `isSpecializationOf` (linking domains hierarchically).

The resulting knowledge graph is substantially larger than the original Mathlib library: starting from approximately 213,000 declarations, the materialization produces over 66 million RDF triples with 6.3 million unique subjects, stored in GraphDB. This $\sim 310\times$ expansion in expressiveness is achieved through ontological typing, reasoner-based inference that supplements the graph with logically derived triples, and the rich structure of the SciLib ontology with its 724 OWL classes. The graph is not a direct copy of Mathlib's internal dependency structure as used, for example, in LeanDojo (Hsiang et al., 2025); it is a substantial extension and reinterpretation of formal mathematical statements through a domain ontology that provides semantic context absent from the raw library.

3.3 Multimodal Embedding Layer

Each Mathlib declaration is represented as a multimodal object linking five modalities through a shared URI (Figure 2): Lean 4 formal code, Russian natural language description, English natural language description, LaTeX symbolic notation, and visual formula rendering. Geometrically, a multimodal object is modeled as a simplex in the embedding space, whose vertices correspond to

Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

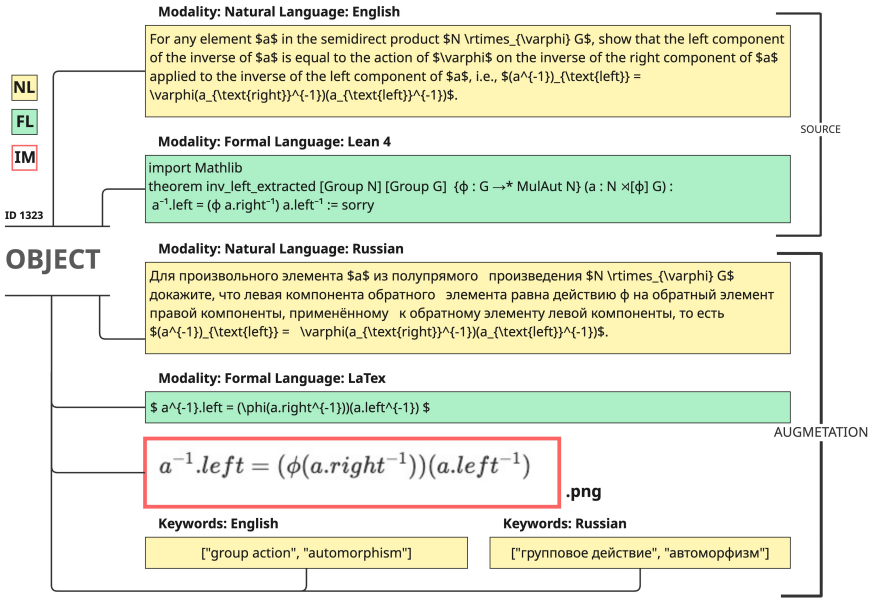

Figure 2: Multimodal representation of a mathematical object. A single Mathlib declaration is linked across five modalities – formal code, Russian and English text, symbolic notation, and visual rendering – through a shared URI, enabling both structured graph traversal and semantic similarity search in a unified vector space.

modalities; the centroid

$$\mu = \frac{1}{M} \sum_{i=1}^{M} z_i \tag{2}$$

serves as an aggregated representation invariant to specific modality (Feng et al., 2014). A dedicated cross-modal encoder, sciLibRuMath, trained on an original dataset of 400,000 mathematical objects, aligns all five modalities into a unified vector space using a combined loss that includes alignment (minimizing simplex volume), contrastive (using centroids as reference vectors with random modality masking), and regularization components. The resulting embedding layer contains over 1 million embeddings across the five modalities, stored in Qdrant, achieving cross-modal Recall@1 above 90% among 10,000 candidates when searching via centroids for most modality pairs. For the experiments in this paper, the vector retrieval mode B1 queries this embedding layer via semantic similarity search, while the graph RAG modes (C11–C23) traverse the RDF knowledge graph directly.

4 Context Delivery Modes

All modes share the same base model (DeepSeek-Prover-V2-7B) and generation parameters. They differ only in how the prompt is constructed before proof generation. We evaluate 8 non-chain-of-thought modes organized in three groups: one baseline, one vector RAG mode, and six graph RAG variants.

4.1 Baseline and Vector RAG

The baseline mode **A0** presents the raw Lean 4 goal to the model with no augmentation – only the theorem statement, necessary imports, and a `sorry` placeholder. This measures the model's parametric knowledge alone. Mode **B1** augments the prompt with lemmas retrieved via semantic search: the model generates candidate Mathlib identifiers (up to 20 LLM calls), each of which

Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

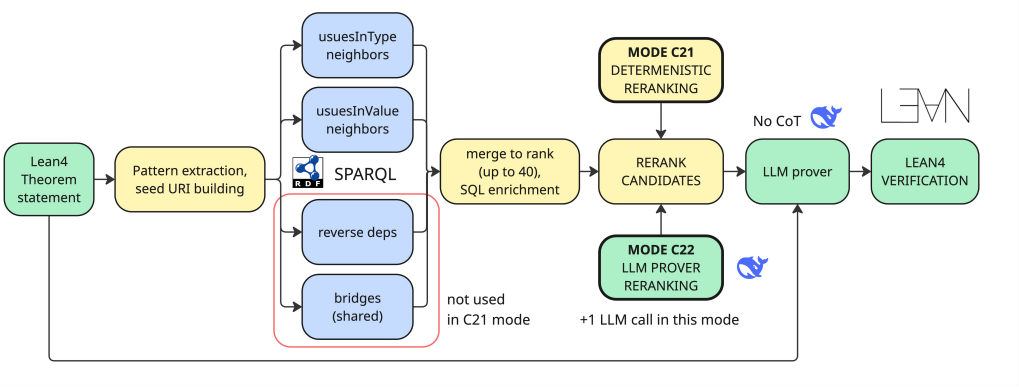

Figure 3: Context delivery pipelines for the two co-leading modes. **C21** (top) combines pattern-based graph expansion with B1 vector retrieval (hybrid). **C23** (bottom) collects candidates from four graph strategies and applies a single LLM reranking call to select the top-8 hints. Both achieve statistically indistinguishable results (p=0.95).

is embedded and queried against the Qdrant collection via the sciLibRuMath encoder. The top-5 nearest neighbors by cosine similarity are appended to the prompt as unstructured hints before the code fence.

4.2 Graph RAG Variants

Six graph RAG modes use the Mathlib dependency graph for context retrieval. They differ along three dimensions: *seed selection* (how graph entry points are chosen), *expansion strategy* (how the graph is traversed), and *vector augmentation* (whether semantic search supplements the graph hints).

Two seed selection approaches are compared. *Model seeds* (modes C1, C2) use an LLM call to generate candidate Mathlib identifiers from the goal text, requiring approximately 30 LLM calls and 130+ seconds per task. *Pattern seeds* (modes C11, C21, C22, C23) use nine regex rules that match goal features – such as inequality operators, `Finset` operations, divisibility predicates, and modular arithmetic symbols – to deterministic Mathlib entry points, requiring zero LLM calls and adding less than one second of overhead. The full list of pattern categories and their mappings is given in Appendix **??**.

From the selected seeds, several expansion strategies are evaluated. **C1** expands model-generated seeds via `usesInType` and `usesInValue` edges. **C11** performs the same typed expansion from pattern seeds. **C22** detects bridge nodes – declarations sharing dependencies with the seed set – identifying structurally related lemmas that may not be direct neighbors. **C23** collects candidates from all four strategies (reverse dependencies, bridges, type neighbors, value neighbors), then applies a single LLM reranking call (64 tokens, greedy decoding) to select the top-8 most relevant hints. Two hybrid modes combine graph and vector retrieval: **C2** merges C1 graph hints with B1 semantic hints (top-3 from each source), and **C21** merges C11 graph hints with B1 semantic hints, leveraging both structural and semantic relevance.

All graph RAG modes format retrieved declarations into four categories reflecting their typical usage in Lean 4 proofs: `apply`/`exact` lemmas applicable as direct proof steps, `rw` lemmas for rewriting equalities, `simp` lemmas for the simplification tactic, and `def` definitions providing vocabulary. This classification makes the retrieved context *actionable*: the model receives not just relevant lemmas, but guidance on *how* to use them. The classification derives from the edge types in the knowledge graph – `usesInType` edges suggest definitional relevance, while `usesInValue` edges suggest proof-body usage. Figure 3 illustrates the pipelines for the two strongest modes.

Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

Table 1: Main results on miniF2F-Test. **All tasks** (197): differences are small and not significant. **Hard tasks** (109, A0 pass@1 $\leq 25\%$): all augmentation modes significantly outperform A0.

	All tasks (197)		Hard tasks (109)			
Mode	pass@1	pass@8	pass@1	Δ A0	p-value	pass@8
A0	41.0	53.8	3.56	–	–	16.5
B1	41.0	55.8	7.34	+3.78	0.0016	20.2
C1	40.9	54.3	6.08	+2.52	0.0135	18.4
C11	42.0	52.0	8.91	+5.32	0.0050	16.7
C2	41.0	54.6	8.10	+4.51	0.0022	19.4
C21	**43.3**	54.1	**10.30**	**+6.71**	**0.0005**	19.4
C22	43.6	52.0	8.68	+5.09	0.0075	15.7
C23	**43.4**	**54.6**	**10.42**	**+6.83**	**0.0011**	**21.3**

5 Experimental Setup

We evaluate on miniF2F-Test (Zheng et al., 2022), a cross-system benchmark of 197 formally stated mathematical problems drawn from AMC, AIME, IMO, and MATH competitions, all formalized in Lean 4. As the base model, we use DeepSeek-Prover-V2-7B (Yin et al., 2025) in non-quantized bf16 precision. Generation parameters are held constant across all modes: temperature T=0.6, top_p=0.95, top_k=40, max_new_tokens=8192, with each task attempted K=8 times per mode.

We define **hard tasks** as those where the unaugmented baseline A0 achieves pass@1 $\leq 25\%$ (at most 2 out of 8 attempts succeed). This yields 109 hard tasks out of 197 total (55%). The threshold is not arbitrary: it identifies tasks where the model's parametric knowledge alone is insufficient. Crucially, it correlates with recognized mathematical difficulty – 91% of AIME problems (10/11), 84% of IMO problems (16/19), and 76% of AMC12 problems (26/34) fall into this category, while only 34% of MATH-D textbook exercises (36/105) do. The threshold thus acts as a principled, model-grounded proxy for mathematical difficulty.

All pairwise mode comparisons use the Wilcoxon signed-rank test (paired, one-sided for directional claims, two-sided for the full matrix in Appendix **??**), computed over per-task pass@1 rates on 109 hard tasks as paired observations. Confidence intervals use the Wilson score method at 95%. Generated proofs are verified by Lean 4 via a dedicated Kafka-based verification service, with each proof compiled against the full Mathlib library using `maxHeartbeats 0`. Results are stored in PostgreSQL with full provenance: attempt ID, mode, timing, raw output, and verification status.

6 Results

6.1 Main Results

Table 1 presents the primary results across all 8 modes, stratified by task difficulty. On all 197 tasks, differences between modes are small (41.0–43.6% pass@1) and not statistically significant. The picture changes dramatically on hard tasks: every augmentation mode significantly outperforms the baseline A0 (all $p < 0.05$), with the best modes nearly tripling the baseline success rate.

6.2 Asymmetric Effect

The central finding is a strongly asymmetric effect (Figure 4). On all tasks, improvements range from $+0.0$ to $+2.1$ percentage points and are not statistically significant ($p > 0.05$). On hard tasks, the same modes yield $+2.5$ to $+6.8$ percentage points, all highly significant ($p < 0.05$). This pattern is consistent with the retrieval-augmentation hypothesis: external knowledge is most valuable when the model's parametric knowledge is insufficient. On easy tasks, the model already has adequate knowledge; additional context is either redundant or occasionally distracting. On hard tasks – predominantly competition-level mathematics – the structured hints from Mathlib's dependency graph provide the missing vocabulary of relevant tactics and lemmas that the model cannot recover from its parameters alone.

Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

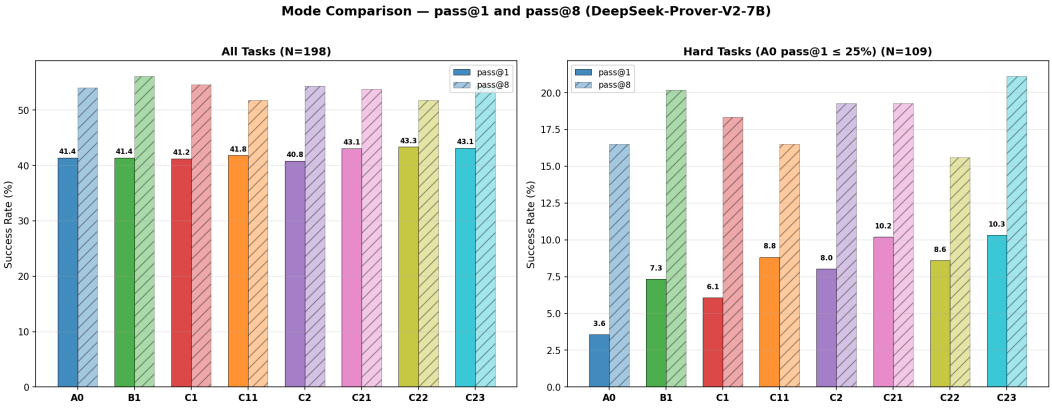

Figure 4: Pass@1 and pass@8 across all modes, stratified by task difficulty. *Left*: on all 197 tasks, modes are indistinguishable. *Right*: on 109 hard tasks, graph RAG modes (C11–C23) substantially outperform the baseline A0. The asymmetry is the key finding.

6.3 Pattern Seeds vs. Model Seeds

Regex-based pattern seeds (C11) achieve pass@1 of 8.91% on hard tasks, significantly outperforming LLM-generated model seeds (C1: 6.08%) with p=0.05. This result is striking given the cost differential: C11 requires zero LLM calls and runs in 12.1 seconds on average, while C1 requires approximately 30 LLM calls and 133.7 seconds ($11\times$ slower). The pattern seeds use nine regex rules matching structural goal features (inequality operators, `Finset` operations, modular arithmetic symbols, and others) to deterministic Mathlib entry points. Their superiority suggests that domain-specific symbolic rules outperform neural text generation for the specific task of selecting entry points into a structured knowledge graph – a finding with implications for the design of retrieval systems in formal mathematics.

6.4 Co-Leaders: C21 and C23

The two strongest modes – C21 and C23 – achieve nearly identical hard-task pass@1 (10.30% vs. 10.42%) through fundamentally different mechanisms. C21 combines pattern-based graph expansion with B1 semantic search, providing both structural and semantic context; the added vector retrieval component accounts for its longer running time (38.4s vs. 12.1s for C11 alone). C23 collects candidates from all four graph strategies, then applies a single LLM reranking call (64 tokens) to select the top-8 hints, trading one cheap LLM call for a more curated context.

The Wilcoxon test between C21 and C23 yields p=0.95 – they are statistically indistinguishable. The convergence of two independent retrieval strategies to the same performance level strengthens the finding: the effect is robust to the specific graph traversal algorithm and likely reflects the intrinsic value of the underlying Mathlib dependency structure. Notably, C21 has stronger statistical significance against A0 (p=0.0005 vs. 0.0011), producing more consistent per-task improvements, while C23 has better pass@8 (21.3% vs. 19.4%) and is $3\times$ faster, producing more diverse correct proofs.

6.5 Coverage Analysis

Beyond per-task improvement, graph RAG expands the set of solvable tasks. Across all 8 modes combined, 118 out of 197 tasks (59.9%) are solved at least once, compared to 106 (53.8%) for A0 alone. Seven tasks are solved exclusively by graph RAG modes (not by A0 or B1), while only two tasks are solved exclusively by the baseline. This asymmetry in exclusive coverage indicates that graph-augmented retrieval unlocks genuinely new problem-solving capabilities rather than merely increasing the probability of solving already-solvable tasks.

Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

7 Case Study: Product Inequality

We illustrate the mechanism of graph-augmented proof generation through a detailed analysis of a representative hard task: `induction_prod1p1onk3le3m1onn`. This task asks to prove that for all positive integers n:

$$\prod_{k=1}^{n}\left(1+\frac{1}{k^3}\right) \le 3-\frac{1}{n} \tag{3}$$

The Lean 4 formalization is:

```
theorem induction_prod1p1onk3le3m1onn
  (n : ℕ) (h₀ : 0 < n) :
 ∏k ∈Finset.Icc 1 n,
   (1 + (1:ℝ) / k^3) ≤(3:ℝ) - 1 / ↑n := by
 sorry
```

This is a non-trivial inductive inequality involving a finite product over cubic reciprocals. The difficulty lies in the inductive step: splitting the product

$$\prod_{k=1}^{n+1}\left(1+\frac{1}{k^3}\right) = \prod_{k=1}^{n}\left(1+\frac{1}{k^3}\right)\cdot\left(1+\frac{1}{(n+1)^3}\right) \tag{4}$$

and proving that the resulting multiplicative inequality closes correctly with field arithmetic. The contrast between modes is stark: A0 fails all 8 attempts (0/8), while graph-augmented modes succeed up to 75% of the time (C22: 6/8, C21: 5/8, C11: 5/8).

The C21 pipeline extracted pattern seeds from the goal – matching finite product, inequality, and division-by-powers features – and expanded them via `usesInType` and `usesInValue` edges. The structured hints include rewrite lemmas such as `div_le_iff₀` and `le_div_iff₀`, which signal multiplicative inequality manipulation patterns; apply lemmas such as `sq_nonneg` and `le_of_eq`, supporting positivity and equality-based steps; and simp set entries such as `pow_nonneg` and `div_self`, directly used by `field_simp` and `positivity` tactics. The complete hint set is given in Appendix **??**.

The best C21 proof (attempt #57192, verified by Lean 4) uses strong induction, splits the product via `Finset.prod_Icc_succ_top`, and applies `mul_le_mul_of_nonneg_right` to leverage the inductive hypothesis:

```
-- C21 proof (verified):
induction' h₀ with n h₀ ih
-- Base case n=1:
· norm_num
-- Inductive step:
· cases n with
  | zero => contradiction
  | succ n =>
    simp_all [Finset.prod_Icc_succ_top]
    -- KEY: use IH on product prefix
    refine' le_trans
      (mul_le_mul_of_nonneg_right
        ih (by positivity)) _
    cases n with
    | zero => norm_num
    | succ n =>
      field_simp
      refine' le_of_sub_nonneg _
      field_simp; ring_nf; positivity
```

The tactic `mul_le_mul_of_nonneg_right` directly corresponds to the retrieved hints `div_le_iff₀` and `le_div_iff₀`, which signal the multiplicative inequality manipulation pattern. The `field_simp; positivity` closure uses `pow_nonneg` and `div_self` from the simp hint set.

Mathematics & AI, Vol. 1, No. 2, 2026
Special Issue: MathAI 2026 Selected Papers
Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

Without graph hints, all 8 A0 attempts adopt a fundamentally flawed strategy: bounding each factor $(1 + 1/k^3)$ individually by 2, yielding $\prod \leq 2^n$, then attempting to prove $2^n \leq 3 - 1/n$ – which is false for $n \geq 2$. The model lacks the vocabulary to recognize that the proof requires strong induction on the product structure and multiplicative inequality manipulation, not per-factor bounding. This example demonstrates the core value proposition: each hint traces to a specific graph edge, and a user inspecting the failed A0 proof alongside the successful C21 proof can identify *exactly which graph-retrieved lemmas* changed the model's strategy.

8 Discussion

The two strongest modes (C21, C23) add minimal overhead to the proof generation pipeline. C23 requires a single LLM call of 64 tokens with greedy decoding and runs in 12.4 seconds on average – comparable to the unaugmented baseline (11.2s). C21, as a hybrid of graph and vector retrieval, runs in 38.4 seconds (comparable to B1 alone at 38.6s), remaining practical for interactive use. By contrast, model-seed modes (C1, C2) require approximately 30–50 LLM calls and 130–160 seconds, yet achieve *lower* hard-task accuracy, suggesting that inference-time cost and retrieval quality are not positively correlated.

Vector RAG (B1) and graph RAG (C11, C22, C23) solve partially disjoint task sets. The two retrieval paradigms operate on different representations – embeddings for semantic similarity, typed edges for structural dependency – and the 7 graph-exclusive versus 2 baseline-exclusive tasks suggest genuine complementarity. The hybrid modes C2 and C21, which combine both sources, confirm this complementarity: C21 achieves the highest statistical significance against A0 (p=0.0005).

The knowledge graph does not "solve" problems – the model does. Rather, the graph provides a *vocabulary of relevant tactics and lemmas* that the model lacks when relying on parametric knowledge alone. This framing is key to understanding both the effect size and the explainability claim: the user can inspect the retrieved vocabulary and understand why the model changed its strategy. The convergence of C21 and C23 (p=0.95) despite their different mechanisms further suggests that the benefit comes from the graph structure itself, not from any specific retrieval algorithm. This is encouraging for practitioners: the choice of expansion strategy matters less than the decision to use the graph at all.

9 Limitations

Our study has several limitations. The sample size of K=8 attempts per task limits per-task statistical power, though results on hard tasks remain robust with 109 paired observations yielding $p < 0.05$ for all modes against A0. Our results are specific to a single model, DeepSeek-Prover-V2-7B; the training-free nature of the approach makes replication with other models straightforward but currently untested. The nine pattern seed categories cover common mathematical structures (inequalities, divisibility, finite sums), but tasks with unusual structures such as combinatorics or geometry may not benefit, and extending pattern coverage is a clear improvement path. The approach requires a typed dependency graph, currently constructed from Lean 4 and Mathlib only; generalization to other proof assistants (Coq, Isabelle) would require analogous graph construction. Finally, we exclude modes with Lean error feedback (R-modes); combining graph RAG with retry loops is a promising but untested configuration.

10 Conclusion

We have shown that structured knowledge retrieval from Mathlib's dependency graph significantly improves neural theorem proving on hard tasks – those where the model's parametric knowledge is insufficient. On 109 hard tasks from miniF2F-Test, the best graph-augmented mode (C23) nearly triples the baseline success rate (pass@1: $3.56\% \rightarrow 10.42\%$, $+6.83$ pp, p=0.001), while adding negligible computational overhead (one LLM call, 12.4 seconds).

Three findings stand out. The effect is strongly asymmetric: graph augmentation helps substantially on hard tasks ($+6.8$ pp, $p<0.001$) while remaining neutral on easy tasks, constraining where retrieval augmentation is useful. Pattern seeds outperform model seeds: deterministic regex-based

Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

entry points are $11\times$ faster and more accurate than LLM-generated ones, demonstrating that simple interpretable rules beat neural generation for knowledge graph navigation. The approach is training-free and composable: no model modification is required, and applying graph-augmented context delivery on top of stronger base models (Lin et al., 2025; Wu et al., 2025) is a natural next step.

The key to explainability is traceability: every hint in the augmented prompt maps to a specific edge in the Mathlib dependency graph. The user can inspect which lemmas were retrieved, through which graph path, and how they influenced the generated proof. This transforms neural theorem proving from a black box into a system where human mathematicians can verify not just the proof, but the reasoning behind its construction.

REFERENCES

J. Asher, Asher, J. (2025). LeanExplore: A search engine for Lean 4 declarations. *arXiv preprint arXiv:2506.11085*.

A. Brack, A. Hoppe, M. Stocker, S. Auer, and R. Ewerth, Brack, A. and Hoppe, A. and Stocker, M. and Auer, S. and Ewerth, R. (2020). Requirements analysis for an open research knowledge graph. *arXiv preprint arXiv:2005.10334*.

S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaëtano, and M. Kohlhase, Buswell, S. and Caprotti, O. and Carlisle, D. and Dewar, M. and Gaëtano, M. and Kohlhase, M. (2004). The OpenMath standard (version 2.0). Technical report, The OpenMath Society.

ByteDance Seed AI4Math, ByteDance Seed AI4Math (2025). Seed-Prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*.

Q. Chen, M. Yang, et al., Chen, Q. and Yang, M. and others (2025). AI4Research: A survey of artificial intelligence for scientific research. *arXiv preprint arXiv:2507.01903*.

Leonardo de Moura and Sebastian Ullrich, de Moura, Leonardo and Ullrich, Sebastian (2021). The Lean 4 theorem prover and programming language. volume and number, Springer.

F. Feng, X. Wang, and R. Li, Feng, F. and Wang, X. and Li, R. (2014). Cross-modal retrieval with correspondence autoencoders. pp., 7–16, ACM.

G. Gao, H. Ju, J. Jiang, Z. Qin, and B. Dong, Gao, G. and Ju, H. and Jiang, J. and Qin, Z. and Dong, B. (2024). A semantic search engine for Mathlib4. *arXiv preprint arXiv:2403.13310*.

G. Gao, Y. Wang, J. Jiang, Q. Gao, Z. Qin, T. Xu, and B. Dong, Gao, G. and Wang, Y. and Jiang, J. and Gao, Q. and Qin, Z. and Xu, T. and Dong, B. (2025). Herald: A natural language annotated Lean 4 dataset. In *International Conference on Learning Representations (ICLR)*.

William A. Howard, Howard, William A. (1980). The formulae-as-types notion of construction. pp., 479–490. Academic Press.

R. Hsiang, W. Adkisson, R. J. George, and A. Anandkumar, Hsiang, R. and Adkisson, W. and George, R. J. and Anandkumar, A. (2025). LeanDojo-v2: A comprehensive library for AI-assisted theorem proving in Lean. In *NeurIPS 2025 Workshop on MATH-AI*.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, et al., Lewis, P. and Perez, E. and Piktus, A. and Petroni, F. and Karpukhin, V. and Goyal, N. and Küttler, H. and Lewis, M. and Yih, W. and Rocktäschel, T. and others (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. pp., 9459–9474.

Y. Lin, Z. Chen, H. Wang, Z. Li, J. Song, et al., Lin, Y. and Chen, Z. and Wang, H. and Li, Z. and Song, J. and others (2025). Goedel-Prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*.

Q. Liu, X. Zheng, R. Xia, X. Qi, Q. Cao, and J. Yan, Liu, Q. and Zheng, X. and Xia, R. and Qi, X. and Cao, Q. and Yan, J. (2025a). Beyond theorem proving: Formulation, framework and benchmark for formal problem-solving. *arXiv preprint arXiv:2505.04528*.

Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

X. Liu, K. Bao, J. Zhang, Y. Liu, Y. Chen, Y. Liu, Y. Jiao, and T. Luo, Liu, X. and Bao, K. and Zhang, J. and Liu, Y. and Chen, Y. and Liu, Y. and Jiao, Y. and Luo, T. (2025b). ATLAS: Autoformalizing theorems through lifting, augmentation, and synthesis of data. *arXiv preprint arXiv:2502.05567*.

Z. Liu, C. He, K. Zheng, M. Hu, et al., Liu, Z. and He, C. and Zheng, K. and Hu, M. and others (2025c). FormalMATH: Benchmarking formal mathematical reasoning of large language models. *arXiv preprint arXiv:2505.08210*.

(1984). *Intuitionistic Type Theory*. Bibliopolis, Naples.

O. A. Nevzorova, N. Zhiltsov, A. Kirillovich, and E. Lipachev, Nevzorova, O. A. and Zhiltsov, N. and Kirillovich, A. and Lipachev, E. (2014). OntoMathPRO ontology: A linked data hub for mathematics. pp., 105–119. Springer.

S. Peroni and D. Shotton, Peroni, S. and Shotton, D. (2018). The SPAR ontologies. pp., 119–136. Springer.

V. A. Serebryakov and O. M. Ataeva, Serebryakov, V. A. and Ataeva, O. M. (2021). Ontology-based approach to modeling of the subject domain "mathematics" in the digital library. *Lobachevskii Journal of Mathematics*, 42:1920–1934.

Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, M. Zhang, Y. Li, Y. Wu, and D. Guo, Shao, Z. and Wang, P. and Zhu, Q. and Xu, R. and Song, J. and Zhang, M. and Li, Y. and Wu, Y. and Guo, D. (2024). DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Z. Shen, N. Huang, F. Yang, Y. Wang, et al., Shen, Z. and Huang, N. and Yang, F. and Wang, Y. and others (2025). REAL-Prover: Retrieval augmented Lean prover for mathematical reasoning. *arXiv preprint arXiv:2505.20613*.

The Harmonic Team, The Harmonic Team (2025). Aristotle: IMO-level automated theorem proving. *arXiv preprint arXiv:2510.01346*.

The mathlib Community, The mathlib Community (2020). The Lean mathematical library. *arXiv preprint arXiv:1910.09336*.

S. Varambally, T. Voice, Y. Sun, Z. Chen, R. Yu, and K. Ye, Varambally, S. and Voice, T. and Sun, Y. and Chen, Z. and Yu, R. and Ye, K. (2025). Hilbert: Recursively building formal proofs with informal reasoning. In *The 5th Workshop on Mathematical Reasoning and AI (MATH-AI) at NeurIPS 2025*.

Philip Wadler, Wadler, Philip (2015). Propositions as types. *Communications of the ACM*, 58(12):75–84.

H. Wang, Z. Liu, X. Lin, J. Liu, F. Sung, et al., Wang, H. and Liu, Z. and Lin, X. and Liu, J. and Sung, F. and others (2025). Kimina-Prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*.

World Wide Web Consortium, World Wide Web Consortium (2010). Mathematical markup language (MathML) version 3.0. Technical report, W3C Working Draft.

X. Wu, H. Wang, Y. Lin, J. Song, et al., Wu, X. and Wang, H. and Lin, Y. and Song, J. and others (2025). Leanabell-Prover-V2: Advancing formal theorem proving via curriculum learning and data augmentation. *arXiv preprint*.

K. Yang, G. Poesia, J. He, W. Li, K. Lauter, S. Chaudhuri, and D. Song, Yang, K. and Poesia, G. and He, J. and Li, W. and Lauter, K. and Chaudhuri, S. and Song, D. (2025). Formal mathematical reasoning – a new frontier in AI. *Position paper, ICML*.

Z. Yin, D. Song, H. Xiong, D. Yu, J. Chen, T. Liu, G. Ma, M. Lin, D. Luo, C. Shao, et al., Yin, Z. and Song, D. and Xiong, H. and Yu, D. and Chen, J. and Liu, T. and Ma, G. and Lin, M. and Luo, D. and Shao, C. and others (2025). DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*.

Received: April 30, 2026 Accepted: May 15, 2026 Published: May 22, 2026

Y. Zhang, R. Wu, P. Cai, X. Wang, G. Yan, S. Mao, D. Wang, and B. Shi, Zhang, Y. and Wu, R. and Cai, P. and Wang, X. and Yan, G. and Mao, S. and Wang, D. and Shi, B. (2025). LeanRAG: Knowledge-graph-based generation with semantic aggregation and hierarchical retrieval. *arXiv preprint arXiv:2508.10391*.

K. Zheng, J. M. Han, and S. Polu, Zheng, K. and Han, J. M. and Polu, S. (2022). miniF2F: A cross-system benchmark for formal Olympiad-level mathematics. In *International Conference on Learning Representations (ICLR)*.