LeanNav: Graph Navigation for Agent-Guided Search in Formal Mathematics

Haikang Deng[1*] **Jia-Chen Gu**[1†] **Yuxuan Lu**[2] **Di Wu**[1]
Yihan Xu[1] **Weichen Winston Yin**[3] **Junyi Zhang**[1] **Nanyun Peng**[1]
[1]University of California, Los Angeles
[2]Northeastern University [3]Axiomatic AI

Abstract

Finding a theorem is often the beginning of library search. Applying it may require a constructor, a coercion, or a lemma that establishes its prerequisites. We present LeanNav, which uses library dependencies to learn these supporting relationships and make them available for agent-guided navigation. Reviewed graph neighborhoods provide supervision for an embedder and a context-aware reranker. In automatic search, LeanNav expands dense results to declarations that share a dependency, then ranks the combined pool using the prover's requested operation and relevant types. LeanNav Interactive lets a proving agent inspect declarations, follow dependencies, and explore their consumers in a private search branch. On MathlibMPR, given the same queries and proof context, LeanNav raises top-20 premise recall from 46.9% to 49.1% over LeanSearch v2 and retrieves a complete premise set for 37.7% of problems, up from 35.5%. Across 310 proving problems, LeanNav solves 10 more than LeanSearch v2 with Kimi K3 and nine more with Gemini 3.1 Pro. Interactive increases the gain over compiler-only proving by 39.7% and 42.1%, respectively, compared with LeanSearch v2. Trace analysis shows how agents use dependency structure to inspect supporting interfaces.

1 Introduction

Formal proof construction depends on finding and applying the right library interfaces. A prover may know the mathematical argument while lacking the declaration that expresses one of its steps. Semantic retrieval helps bridge this gap, but a theorem closely matching the query can still require a constructor, coercion, or prerequisite lemma before it applies. Lean 4 (de Moura and Ullrich, 2021) makes these requirements explicit through types and compiler feedback. Library search therefore needs to support both finding a relevant statement and assembling the interfaces for its use.

We use Quiver (Yin, 2026) to obtain a dependency graph of Mathlib (The mathlib Community, 2020), with edges recording references in declaration types and proof bodies. We use these edges to curate training data, expand search results automatically, and let the prover navigate the library. An edge $d \to p$ connects a declaration to a dependency. Reversing the edge exposes consumers of that dependency, and $d \to p \leftarrow h$ identifies declarations sharing an interface. Such relationships can connect a theorem to supporting operations that are difficult to express in a single semantic query.

Our examination of dense retrieval results motivates this distinction. Projecting candidates onto the graph can reveal a neighborhood of constructors and lemmas for one operation, several complementary proof stages, or an isolated specialized adapter. Component size alone does not identify which declaration a proof needs. Likewise, a shared dependency can connect both useful interfaces and general infrastructure. The graph provides candidates and relationships whose relevance must be assessed against the requested operation and its prerequisites.

LeanNav combines these ideas (Figure 1). Reviewed graph relationships train an embedder and a context-aware reranker. During automatic search, LeanNav uses shared dependencies to propose candidates that dense retrieval may overlook. LeanNav Interactive instead gives the agent a private search branch in which it can refine queries, inspect prerequisites, and navigate dependencies and consumers. Both interfaces return selected declarations to the same outer proving loop.

Related work. Learned premise selection connects retrieval to formal reasoning. Magnusham-

*Correspondence: frankdenghaikang@gmail.com
†The six middle authors are listed alphabetically by surname.

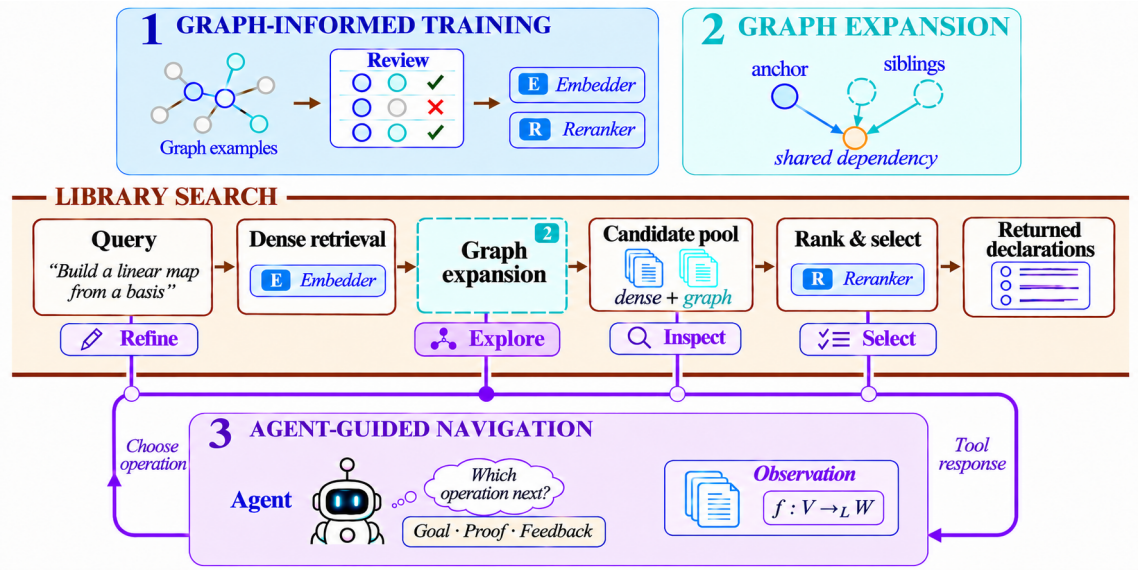


Figure 1: LeanNav uses the same library graph to curate training data (1), expand dense results to sibling declarations (2), and let the agent navigate during search (3).

mer uses contrastive retrieval and reranking in Isabelle (Mikuła et al., 2024). LeanDojo provides premise annotations for retrieval-augmented Lean proving (Yang et al., 2023), while LeanPremise trains selection for LeanHammer (Zhu et al., 2025). Lean Copilot integrates premise selection and proof assistance into Lean (Song et al., 2024). These systems establish the value of choosing premises in a formal context.

Library search supports both formal patterns and informal requests. Loogle searches declaration names and type patterns (Breitner, 2023). Moogle (Morph Labs, 2023) and the original LeanSearch (Gao et al., 2024) support natural-language theorem search. Lean Finder learns retrieval from user intents (Lu et al., 2025). LeanExplore combines semantic, lexical, and PageRank signals (Asher, 2025), and LeanSearch v2 combines dense retrieval with reranking (Gao et al., 2026). LeanGraph-Search propagates relevance along forward dependencies (Xu et al., 2026). We evaluate its native graph as an independent baseline. LeanNav studies reviewed graph relationships as retrieval supervision and exposes dependencies and consumers for agent-directed search, connecting learned relevance with explicit library navigation.

2 Graph-Informed Retrieval Learning

Learning supporting relationships. We generate and review natural-language queries against extracted Mathlib declarations. Qwen3-32B curates the query bank, and Qwen3.8-27B reviews graph-helper proposals (Appendix A). Ordinary retrieval examples pair a query q with an answer declaration d. We supplement them with helper candidates mined from dependencies, reverse consumers, and shared-dependency neighborhoods. Agreement-filtered teacher reviews identify useful supporting interfaces and unrelated contrasts. The encoder learns both to answer the query and to place compatible helpers near the source declaration.

Let s_θ denote embedding similarity. The objective combines a query-to-answer contrastive loss with a declaration-anchored helper loss:

$$\mathcal{L}_E = \mathcal{L}_{\text{answer}} + \lambda \mathcal{L}_{\text{helper}}, \tag{1}$$

$$\ell_h(d, h, n) = \text{softplus}\left(\frac{s_\theta(d, n) - s_\theta(d, h)}{\tau}\right), \tag{2}$$

where h is a reviewed helper and n an unrelated contrast. Coverage normalization accounts for examples lacking helper supervision. The shared encoder subsequently serves ordinary dense queries.

Learning contextual relevance. We adapt Qwen3-Reranker-8B (Zhang et al., 2025) with LoRA (Hu et al., 2022). Training queries come from Kimi and Gemini on training-source theorems. Their candidate pools combine dense results, direct dependencies, and reverse consumers. Assistant-curated review identifies direct matches, compatible supporting interfaces, and incompatible operations. A source-balanced pairwise loss learns these preferences.

Retrieval configuration	nDCG@10	R@10	R@100
Embedding-only retrieval			
Pretrained	49.47	66.67	82.47
Query-only tuning	58.76	75.06	89.59
Graph-informed tuning	**60.71**	**76.42**	**92.02**
Same pretrained reranker, 200 candidates			
LeanSearch v2	62.24	77.90	85.93
LeanNav embedder	**63.43**	**79.88**	**92.55**

Table 1: MathlibQR-810 retrieval (%). Tuned rows are three-seed means. The lower block holds the pretrained reranker fixed and uses query-only input with dense candidates.

Two optional fields express the agent's search intent: the *requested operation* and *relevant types and prerequisites*. For example, the operation may ask to transport a property across an equivalence. The second field identifies the available instances and remaining requirements. Context dropout trains the scorer to handle both enriched and query-only inputs.

LeanNav retrieval. At inference, the agent emits the two context fields alongside each query. LeanNav uses the highest-ranked dense results as anchors. If an anchor uses a declaration p, other declarations that use p are its siblings. An evaluation lemma and an injectivity theorem, for example, can share a constructor even when their statements look quite different. LeanNav adds siblings from selected anchors to the candidate pool. One context-aware reranker scores both sources and returns the ten highest-ranked declarations, with no fixed quota for siblings. Appendix A details the data and objectives.

Standalone results. Table 1 evaluates the 810 MathlibQR queries mapped to the shared corpus. Graph-informed embedding training improves over query-only tuning by 1.95 nDCG@10 points and 2.43 Recall@100 points. Its benefit also persists when the pretrained reranker is held fixed. The tuned results average three training seeds.

MathlibMPR asks whether search recovers the premises of a proof plan. We replay Kimi and Gemini queries on 69 problems, scoring 100 candidates per query. With the same proof context, LeanNav retrieves 49.1% of annotated premise groups in the merged top 20, versus 46.9% for LeanSearch v2 (Table 2). Complete coverage rises from 35.5% to 37.7%.

	Recall		Covered	
Search engine	@10	@20	@10	@20
LeanSearch v2	37.04	44.75	26.81	33.33
+ context	39.05	46.92	**30.43**	35.51
LeanGraphSearch	35.06	42.58	24.64	32.61
+ context	34.89	45.34	26.09	35.51
LeanNav	**39.88**	**49.09**	**30.43**	**37.68**

Table 2: MathlibMPR premise retrieval (%), averaged over the two planners. Covered is the share of problems with a complete annotated premise set in the top k. Baselines marked + context receive the same requested operation and types as LeanNav. Table 4 reports all five cutoffs.

3 Interactive Library Navigation

LeanNav Interactive lets the prover refine retrieval after inspecting a candidate. Its private branch sees the theorem, current proof, compiler feedback, and queries. It applies the principle of interleaving reasoning with tool actions (Yao et al., 2023) to library interfaces.

The agent can issue semantic searches, inspect complete types, follow dependencies, retrieve consumers, and rerank or select known candidates. Exploration is optional, and the final selection can mix dense and graph discoveries freely.

For example, the agent can inspect d, identify a dependency p, and request other consumers h of p. It checks their prerequisites before returning them. Direct dependency traversal and query refinement provide other routes when the current neighborhood lacks a useful interface.

The prover receives selected declarations with available descriptions, signatures, and complete Lean types. Navigation stays private. Proof compilation remains in the outer repair loop.

Evaluation. The proving suite contains 310 problems: 100 FATE-H, 100 FATE-X, 50 MathlibMPR-Prop, and 30 each from IMO Basic and Advanced. Both agents use HIGH reasoning and a 65,536-token output cap for main and query generations. An initial proof attempt is followed by up to eight repairs. Each search round generates five queries and returns ten declarations per query, deduplicated across the batch. We count complete, compiler-verified proofs.

LeanNav and LeanNav Interactive use the same trained retrieval models. The automatic system allocates 50 scoring pairs per query. Interactive allocates the same total of 250 across the five queries

Search engine	FATE-H /100	FATE-X /100	MPR-Prop /50	IMO /60	**Total /310**	Est. cost USD/prob.
Kimi K3, HIGH reasoning						
Compiler only	26	9	17	**24**	76	$2.33
LeanSearch v2	56	18	38	22	134	$2.80
LeanGraphSearch	58	19	36	**24**	137	$2.81
LeanNav	61	20	41	22	144	$2.69
LeanNav Interactive	**66**	**26**	**42**	23	**157**	$5.31
Gemini 3.1 Pro, HIGH reasoning						
Compiler only	23	7	13	15	58	$2.02
LeanSearch v2	41	12	28	15	96	$2.13
LeanGraphSearch	44	12	27	13	96	$2.14
LeanNav	**49**	13	27	16	105	$2.11
LeanNav Interactive	46	**15**	**32**	**19**	**112**	$4.25

Table 3: Verified proofs on the released suite and mean estimated API cost per executed problem.

in a repair round. Both return up to ten declarations per query. Interactive also uses a temporary agent session to choose and inspect candidates. Appendix C specifies the baseline adaptation and search allowances.

4 Analysis and Discussion

Where the improvements occur. LeanNav solves 144 problems with Kimi and 105 with Gemini, compared with 134 and 96 for LeanSearch v2 (Table 3). The largest gain for both agents is on FATE-H: five more problems for Kimi and eight for Gemini. Interactive solves a further 13 problems with Kimi and seven with Gemini overall. Its largest gains over automatic search occur on FATE-X for Kimi (20 to 26) and MathlibMPR-Prop for Gemini (27 to 32). On Gemini FATE-H, automatic search remains ahead (49 versus 46). We measure *search-tool lift* as the increase in solved problems over compiler-only proving. Moving from LeanSearch v2 to Interactive raises this lift from 58 to 81 problems for Kimi and from 38 to 54 for Gemini.

What agents explore. With pretrained retrieval models, both agents more often explored declarations that use a selected declaration than followed its dependencies: 675 versus 64 times for Kimi and 348 versus 27 for Gemini.

Structural navigation in practice. Which graph discoveries reach the proof itself? In 47 of 157 accepted Kimi proofs (29.9%) and 15 of 112 Gemini proofs (13.4%), the compiled proof uses a theorem that Interactive first found through the graph and returned to the prover. These theorems lay outside all five initial dense pools in the round they were first returned.

Interaction, graph access, and cost. On the current suite, mean estimated cost per executed problem is $2.69 for automatic LeanNav versus $5.31 for Interactive with Kimi, and $2.11 versus $4.25 with Gemini. Interactive uses an additional agent session to inspect candidates before returning declarations (Table 3).

Conclusion. A theorem can match a query even when the prover still needs a lemma to apply it. LeanNav learns from Mathlib's dependencies and uses them to find supporting declarations beyond dense matches. The operation and types in the current proof guide which candidates it returns. On MathlibMPR, this approach recovers complete premise sets more often than LeanSearch v2 given the same proof context. Letting the prover inspect candidates and choose where to search next leads to more verified proofs for both Kimi and Gemini than automatic search alone. Some of those proofs use a theorem that was absent from the initial dense results and surfaced through the graph.

References

Justin Asher. 2025. LeanExplore: A search engine for Lean 4 declarations. *Preprint*, arXiv:2506.11085.

Joachim Breitner. 2023. Loogle: A search tool for Lean and Mathlib. Software.

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

Guoxiong Gao, Haocheng Ju, Jiedong Jiang, Zihan Qin, and Bin Dong. 2024. A semantic search engine for Mathlib4. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8001–8013.

Guoxiong Gao, Zeming Sun, Jiedong Jiang, Yutong Wang, Jingda Xu, Peihao Wu, Bryan Dai, and Bin Dong. 2026. LeanSearch v2: Global premise retrieval for Lean 4 theorem proving. *Preprint*, arXiv:2605.13137.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Jialin Lu, Kye Emond, Kaiyu Yang, Swarat Chaudhuri, Weiran Sun, and Wuyang Chen. 2025. Lean Finder: Semantic search for mathlib that understands user intents. *Preprint*, arXiv:2510.15940.

Maciej Mikuła, Szymon Tworkowski, Szymon Antoniak, Bartosz Piotrowski, Albert Qiaochu Jiang, Jin Peng Zhou, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. 2024. Magnushammer: A transformer-based approach to premise selection. In *International Conference on Learning Representations*.

Morph Labs. 2023. Moogle: Semantic search over Mathlib4. Software.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. 2024. Lean Copilot: Large language models as copilots for theorem proving in Lean. *Preprint*, arXiv:2404.12534.

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

Yihan Xu, Jia-Chen Gu, Junyi Zhang, Di Wu, Haikang Deng, Po-Nien Kung, Kai-Wei Chang, Amit Sahai, Wei Wang, and Nanyun Peng. 2026. Lean Graph Search for formal theorem proving. *Preprint*, arXiv:2610.22339.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. 2023. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, volume 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Weichen Winston Yin. 2026. Quiver: A knowledge-graph extraction pipeline for Lean libraries. AITP 2026 abstract.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 Embedding: Advancing text embedding and reranking through foundation models. *Preprint*, arXiv:2506.05176.

Thomas Zhu, Joshua Clune, Jeremy Avigad, Albert Qiaochu Jiang, and Sean Welleck. 2025. Premise selection for a Lean hammer. *Preprint*, arXiv:2506.07477.

A Retrieval Training and Data

Query and helper supervision. We extract formal declarations from Mathlib and reuse LeanSearch v2 descriptions. Qwen3-32B generates precise and plain-language requests from source statements and reviews source fidelity and proposed negatives. The retained bank contains 4,559 queries from 3,261 declarations, with seven reviewed negatives per query. Query-only and graph-informed training use this same bank.

Helper proposals come from proof-body dependencies, reverse consumers, and shared-dependency siblings with 2–64 consumers. Qwen3.8-27B reviews 1,536 slates twice with shuffled orders, using the query, full types, and descriptions. Agreement and compatibility checks retain 1,011 helper pairs across 658 queries, of which 653 have usable positive/negative contrasts. Uncertain or conflicting judgments are excluded.

Encoder training. We adapt Qwen3-Embedding-8B with rank-16 LoRA. The helper loss in Section 2 uses $\tau = 0.05$ and coverage-normalized $\lambda = 0.1/(658/4559)$. Examples without helper comparisons contribute only the answer loss.

Context-aware reranker. We adapt Qwen3-Reranker-8B with rank-16 LoRA and a source-balanced pairwise loss, $\mathrm{softplus}(r(n) - r(p))$. The retained corpus has 125 queries from 14 training theorem sources, generated by Kimi and Gemini at failed-proof checkpoints. Each pool contains up to 150 dense, dependency, and consumer candidates. Reference proofs remain outside generation prompts.

The labels are assistant-curated mathematical judgments based on actual queries, context fields, and complete candidate types, with selected applications checked in Lean. These partial labels are separate from the Qwen3.8-27B encoder-helper reviews. Of the 125 records, 54 queries from ten sources provide 120 positive/negative ordering pairs. Unjudged candidates are excluded from supervised comparisons. Training uses judgments valid with and without the optional fields and applies context dropout.

B MathlibMPR Premise Retrieval

LeanSearch v2 with context leads at five results (Table 4). At ten, it ties LeanNav on complete premise coverage. At 20, 30, and 50, LeanNav has the highest recall and complete coverage among the systems compared.

C Proving Protocol

All arms use Lean/Mathlib 4.28.0-rc1 and the same 306 executable problems. The four unrun FATE-H inputs remain in the reported denominator of 310 without solved credit. The loop permits one initial attempt and eight repairs, with five queries per search round and ten returned declarations per query. Returned content includes available descriptions, signatures, and complete types.

Kimi uses `kimi-k3`, HIGH reasoning, temperature 1.0, and `top_p`=0.95. Gemini uses `gemini-3.1-pro-preview`, HIGH thinking, and provider-default sampling. Main and query generations allow 65,536 output tokens, including reasoning. Navigation allows 32,768.

Search configurations. For proving, automatic LeanNav scores 25 dense candidates and up to 25 distinct siblings reached through shared dependencies. If fewer than 25 new siblings are found, further dense results fill the pool to 50 distinct candidates. The ten best contextual scores determine the return. LeanGraphSearch uses pretrained retrieval models, its native Jixia graph, directed PPR, and fusion with reranker scores. Our 50-pair adaptation allocates 25 first-stage pairs and up to 25 additional pairs. LeanNav Interactive uses the same trained models and a shared limit of 250 scoring pairs per five-query round. Its private exploration does not enlarge the final return window.

	Recall					Covered				
Search engine	@5	@10	@20	@30	@50	@5	@10	@20	@30	@50
LeanSearch v2	25.10	37.04	44.75	48.38	54.68	14.49	26.81	33.33	35.51	41.30
+ context	**29.80**	39.05	46.92	53.33	58.82	**22.46**	**30.43**	35.51	41.30	45.65
LeanGraphSearch	24.48	35.06	42.58	46.54	52.77	14.49	24.64	32.61	35.51	41.30
+ context	26.52	34.89	45.34	49.78	55.44	19.57	26.09	35.51	36.96	43.48
LeanNav	29.23	**39.88**	**49.09**	**55.20**	**59.41**	21.74	**30.43**	**37.68**	**43.48**	**46.38**

Table 4: MathlibMPR premise retrieval (%) on 69 problems, averaged over Kimi and Gemini proof plans. Recall measures the fraction of annotated premise groups retrieved. Covered counts problems for which the top k contain a complete annotated premise set.