# Quiver: A Knowledge-Graph Extraction Pipeline for Lean Libraries

Weichen Winston Yin[1]

Axiomatic_AI, Cambridge, MA, USA
winston.yin@axiomatic-ai.com

Quiver indexes a Lean library's full elaborated structure — every declaration, proof state, tactic step, and typeclass-synthesis chain — as a typed knowledge graph and augments it with semantic embedding and a structural similarity graph. From Mathlib, it extracted 7.3M nodes and 50M edges by materialising 399K declarations, 4.5M premises, 720K proof steps, 99K typeclass instances. As an extraction pipeline, it runs as part of a continuous integration workflow on any Lean project. The graph may be interactively traversed by a human via a visual web interface and queried by an LLM agent via an MCP server.

## 1 The library-indexing gap

Tools that index a Lean library either miss what Lean's elaborator produces or expose only a narrow slice of it. *Text-indexed* tools — ripgrep and embedding-based search engines (LeanSearch, LeanExplore [6], Lean Finder [7]) — cannot see typeclass synthesis, notation expansion, coercions, simp-set application, or macro-generated declarations, none of which appear in source. *Elaboration-aware tools with fixed interfaces* — doc-gen4 (a declaration-keyed HTML browser) and Loogle (a user-supplied Lean pattern matcher over declaration types) — see the elaborated form but expose no tactic, instance-synthesis, or step-level structure, and no compositional query language. `lean-lsp`, an elaboration-aware tool, is built for real-time, single-file, per-cursor queries: rich elaborated state is computed on demand and discarded immediately, never joined across modules. Library-scale questions — *every theorem sharing hypotheses $P$ and $Q$* — require a persistent index that holds the elaborated state of every module at once. Finally, existing ML-oriented extractors — LeanDojo [1], ntp-toolkit, Pantograph — emit *flat* tactic and premise tuples frozen at a fixed Mathlib snapshot, leaving structural reconstruction to the consumer and going stale as the library evolves. A further limitation cuts across these categories: many existing tools (LeanSearch, Moogle, Lean Finder, the published LeanDojo dataset) are deployed only for Mathlib, with no path to a user's own Lean project.

## 2 Pipeline and schema

Extraction elaborates each module and captures the declarations themselves, the decomposition of their statements into premises and targets, the goal- and tactic-step structure of their proofs, the typeclass-synthesis chains discovered along the way, and the surrounding module structure. Each type expression is content-addressed by a canonical identifier, so that definitionally equal types collapse to a single *condition*; conditions shared across many declarations are the queryable join points for cross-cutting searches. Cross-references between declarations, the action of each tactic step on goal states, the slots it consumes and creates, and the provenance of each synthesised instance are all recorded as typed edges. The resulting graph is stored in Neo4j. On top of it we build a natural-language embedding index and a precomputed *least-general-generalisation* (LGG, anti-unification [10]) similarity graph. LGG returns the shared structural skeleton of two terms; we derive a similarity score in $[0, 1]$ from the size of the skeleton relative to the inputs and build the approximate-$k$NN graph with NN-Descent [11].

# 3 Contributions

**1. Canonical exact-match retrieval.** Condition IDs are hashes of canonicalised kernel expressions; two hypotheses with the same canonical type share *the same* condition ID. Existing structural retrievers operate on similarity scores [4, 5] or user-supplied patterns (Loogle); none offer $O(1)$ exact-match lookup on canonical term identity.

**2. Unified queryable knowledge graph.** Cross-cutting queries —*every step that uses lemma L through any instance-synthesis path*, *every theorem whose hypothesis matches this canonical condition*—are joins, not separate extraction passes. ntp-toolkit and Pantograph emit similar primitives as flat JSONL streams; reconstructing the join is the consumer's problem.

**3. Step-level provenance as a graph object.** Each tactic step records the slots it consumes and creates. These edges are queryable library-wide: *every step in Mathlib that creates a slot of this canonical shape*, *every proof that consumes premise $X$ at step $k$*. Other extractors record this data per theorem.

**4. Instance synthesis as a queryable graph.** Typeclass-instance synthesis chains—which instance was synthesised by which provider, under which slot—are recorded as graph edges. Lean prints these per-elaboration; no other tool we know of materialises them as a library-wide queryable structure.

# 4 Capabilities enabled by graph cross-cutting

**Structural-pattern discovery.** The precomputed LGG similarity graph turns a single-lemma query into a structural-pattern query across the whole library. Starting from `Finset.prod_le_one`, whose conclusion is $\prod_{i\in s} f(i) \le 1$, Quiver returns LGG neighbours that share its overall shape, including `Finset.sum_nonpos` ($\sum_{i\in s} f(i) \le 0$), `Nat.ModEq.prod_one` ($\prod_{x\in s} f(x) \equiv 1 \pmod n$), and `MeasureTheory.integral_eq_zero_of_ae` ($\int_\alpha f \,\mathrm{d}\mu = 0$). These four theorems live in three different subjects and use three different operators ($\prod$, $\sum$, $\int$), yet LGG recovers them as instances of a single structural schema. Unlike Loogle, which requires a user-supplied pattern, LGG derives the shared pattern from the anchor itself.

**Premise-tuple context signatures.** Since Quiver indexes premises by canonical condition ID derived from the underlying kernel expression, a simple graph join query completely enumerates *which declarations share a given set of premises*, regardless of how these premises may be spelt in the code (such as with different variable names).

For example, the condition `IsOpen v` appears in 599 Mathlib declarations and `CompactSpace` $\beta$ in 576, where `v` and $\beta$ stand for any matching variables in the code. Their intersection contains exactly nine declarations, spanning six modules across topology and dynamics — precisely Mathlib's inventory of what can be said about *an open set in a compact space*, none of which a neural premise retriever trained on a frozen snapshot can guarantee to recall.

# 5 Position and status

Quiver complements existing Lean indexers — LeanDojo [1], MMLKG [2], Li et al. [3], structural premise selectors [4, 5], and retrieval engines [6, 7] — and premise-selection systems (Magnushammer [9], LeanHammer [8]) by providing infrastructure to generate datasets in both forms: a queryable database for direct symbolic access, and structured training data for neural premise selectors. The extraction binary, schema, web UI, and MCP server are functional and the snapshot is reproducible; a pilot Mathlib refactor study is in progress. Planned evaluation covers refactor-impact estimation, declaration retrieval by premise tuple, NL retrieval accuracy, and LLM-agent token-cost reduction on library-management tasks. We commit to a public dataset release of the extracted Mathlib graph.

# References


[1] K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, and A. Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *NeurIPS Datasets and Benchmarks Track*, 2023.

[2] D. Tomaszuk, Ł. Szeremeta, and A. Korniłowicz. MMLKG: Knowledge graph for mathematical definitions, statements and proofs. *Scientific Data*, 10:791, 2023.

[3] X. Li, N. Peng, S. Severini, and P. Shafto. The network structure of Mathlib. arXiv:2604.24797, 2026.

[4] Z. Wang, A. Dong, and Z. Wen. Tree-based premise selection for Lean4. In *NeurIPS*, 2025.

[5] J. Petrovčič, D. E. Narvaez Denis, and L. Todorovski. Combining textual and structural information for premise selection in Lean. arXiv:2510.23637, 2025.

[6] J. Asher. LeanExplore: A search engine for Lean 4 declarations. arXiv:2506.11085, 2025.

[7] H. Lu and S. Emond. Lean Finder: Semantic search for Mathlib that understands user intents. In *ICLR*, 2026.

[8] T. Zhu, J. Clune, J. Avigad, A. Q. Jiang, and S. Welleck. Premise selection for a Lean hammer. arXiv:2506.07477, 2025.

[9] M. Mikuła, S. Tworkowski, et al. Magnushammer: A transformer-based approach to premise selection. In *ICLR*, 2024.

[10] G. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5, 1970.

[11] W. Dong, C. Moses, and K. Li. Efficient $k$-nearest neighbor graph construction for generic similarity measures. In *WWW*, 2011.