DAPS: Dependency-Aware Premise Selection for LLM Theorem Proving

Yinya Huang [*1 2] **Zixuan Chen** [*2] **Wenyuan Jiang** [2] **Junling Wang** [1 2] **Mrinmaya Sachan** [2]

Abstract

LLM-based theorem proving in Lean 4 has advanced rapidly, but provers are difficult to leverage newly contributed lemmas. Existing neural selectors rank candidates by semantic similarity, with three intrinsic limitations: frequency-skewed retrieval, isolated pairwise ranking, and degradation on out-of-distribution queries such as competition problems and informal language. Given that mathematics is structured by dependencies rather than similarities, we restore this signal to both the training data and the model. We conduct the first systematic mining of 53 Lean Blueprint projects, a previously untapped corpus of 3,805 manually curated informal+formal nodes that capture rare, research-level premises absent from Mathlib4. We further pair it with the extraction of comprehensive, typed, multi-level Mathlib4 dependencies. We then introduce **DAPS**, a **d**ependency-**a**ware **p**remise **s**elector with a structural neighborhood encoder, a group-level contrastive objective, and a Mathlib-then-Blueprint adaptation procedure. **DAPS** reaches Recall@32 of 89.31 on Mathlib4-Heldout, 11.47 over the strong selector LeanHammer, and holds its lead across out-of-distribution benchmarks. **DAPS** further improves three general-purpose LLMs on the informal IMO-ProofBench. Dependency graphs, benchmarks, and code are available[1] and will be released under a permissive license upon publication.

[1]ETH AI Center, ETH Zürich [2]Department of Computer Science, ETH Zürich. Correspondence to: Yinya Huang <yinya.huang@ai.ethz.ch>, Mrinmaya Sachan <mrinmaya.sachan@inf.ethz.ch>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

[1]https://anonymous.4open.science/r/DAPS-61CF

1. Introduction

LLM-based Lean 4 theorem proving has progressed rapidly, with frontier provers such as DeepSeek-Prover-V2 (Ren et al., 2025), Kimina-Prover (Wang et al., 2025), and Goedel-Prover-V2 (Lin et al., 2025b) reaching IMO-level performance. Yet a fundamental limitation persists: these provers operate within a self-contained paradigm, generating subgoals and reusing constants they have already memorized. Provers are difficult to leverage newly contributed lemmas in Mathlib4 (The mathlib Community, 2020) or theorems established by domain experts in collaborative formalization projects, because they lack a reliable way to identify the appropriate premises among the hundreds of thousands of available declarations. Closing this gap requires premise selection that can reliably reach the rare and recently introduced lemmas, not just the popular foundational ones the prover has already memorized.

Existing neural selectors (Yang et al., 2023; Zhu et al., 2025; Mikula et al., 2024) frame premise selection as semantic retrieval, a framing with three intrinsic limitations that better encoding cannot fix. (P1) **Frequency-skewed retrieval**: rare and recently added premises with weak text signal are displaced in top-K by popular foundational lemmas the prover already controls. (P2) **Group-level relevance**: pairwise similarity ranks each lemma in isolation, missing whether the top-K cohere as a usable proof toolkit. (P3) **Distribution shift**: Mathlib-trained semantic features degrade on out-of-distribution queries such as competition problems, research blueprints, and informal language. The three share a single root cause. Mathematics is structured by dependencies rather than similarities, so what makes a lemma useful is its position in the dependency graph, for example, the classes it presupposes, the lemmas it composes with, and its distance to known premises. Restoring this structure to both the training data and the model architecture is the unifying principle behind this work.

As shown in Figure 1 with the `flt-regular` project as a running example[2], Lean 4 makes at least two rich de-

[2]https://leanprover-community.github.io/flt-regular/blueprint/dep_graph_document.html

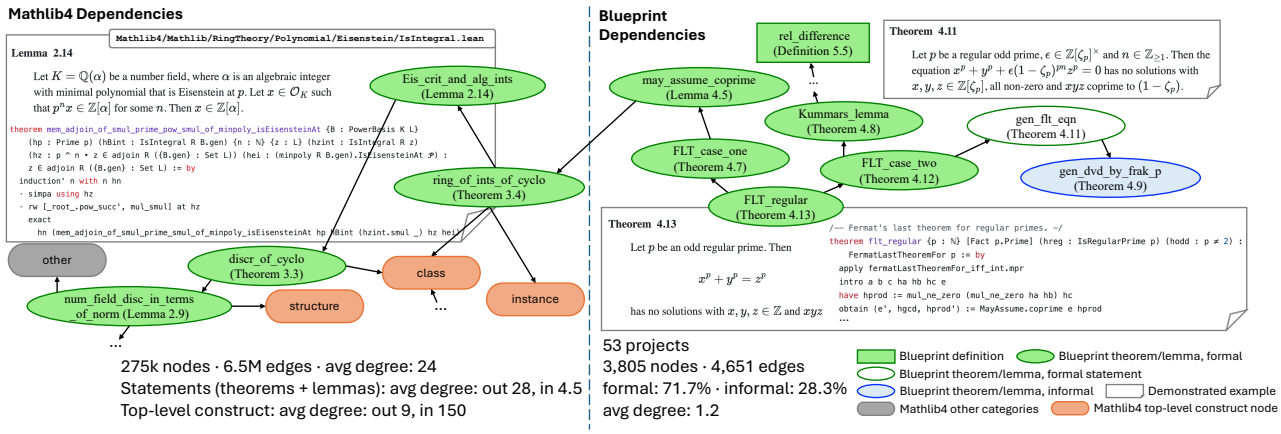


Figure 1. One dependency graph spanning Mathlib4 (left) and a representative Blueprint project (right). The two resources differ in density and composition. Mathlib4 averages 24 edges per node with strong directional asymmetry across node types, while Blueprint dependencies are much sparser, averaging 1.2, but carry research-level out-of-Mathlib4 declarations and informal LaTeX signals.

pendency resources available. **Mathlib4's compiled constant graph** (left) is a typed, heterogeneous DAG with 275k+ nodes and 6.5M edges, where each edge points from a declaration to a constant it uses in its statement or proof. The graph is directionally asymmetric: statement nodes (e.g., `theorem`, `lemma`) have high out-degree and low in-degree, while foundational constructs (e.g., `class`, `structure`, `instance`, `inductive`) show the opposite pattern. **Lean Blueprint projects**[3] (right) are collaborative formalization repositories that contribute 3,805 manually curated nodes (71.7% formal, 28.3% informal) across 53 research projects. Many of these nodes are recent or not-yet-formalized results absent from Mathlib4, so the graph is sparse with an average degree of just 1.2. *Against P1's frequency-skewed retrieval*, Blueprints supply the rare, advanced premises absent from Mathlib4, while the heavy-tailed shape of Mathlib4 itself (top 10% of nodes capture 56% of in-edges) lets us measure the frequency skew directly. *For P2's group-level relevance*, the Mathlib graph encodes which sets of premises are jointly used per proof, providing direct supervision over premise groups rather than individual lemmas. *For P3's distribution shift*, the two resources span in-domain Mathlib (left of Figure 1) and out-of-distribution informal+formal queries (right), and we additionally collect out-of-distribution competition and synthetic proofs as a third source.

We accordingly derive four evaluation benchmarks of increasing distribution shift: Mathlib4-Random (1,000 in-distribution queries with per-declaration random holdout), Mathlib4-Heldout (1,000 in-distribution queries with file-level isolation), **DAPS**-Collection (1,000 queries mined from 152k open-source competitions and synthetic proofs), and **DAPS**-Blueprint (236 informal+formal queries from advanced research projects).

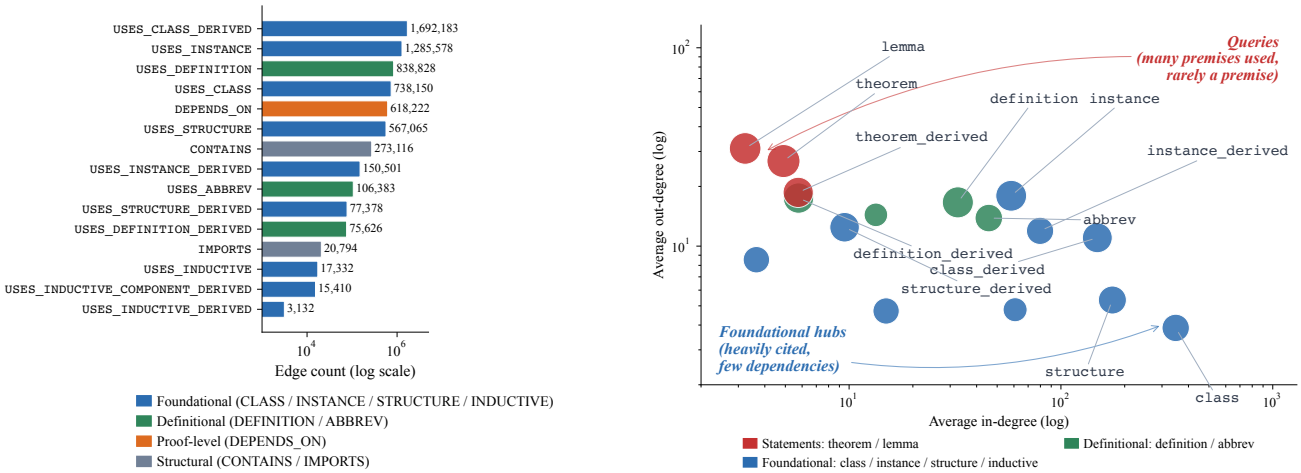
We then introduce **DAPS**, a **d**ependency-**a**ware **p**remise **s**elector that addresses the three problems. *To counter the long-tail collapse of P1*, **DAPS** has a structural neighborhood encoding that augments each candidate's text representation with features summarizing its position in the dependency graph (i.e., the kinds of declarations it depends on, its connectivity, and its relation to the query). *For the group-level relevance of P2*, **DAPS** has a group-level training objective that pulls the multiple gold premises of a query towards the query embedding as a single group, away from negatives, so the model learns to rank jointly-useful premise sets near the top, not just individual nearest neighbors. *To absorb the distribution shift of P3*, **DAPS** has a Mathlib-then-Blueprint adaptation procedure that first trains on the abundant typed Mathlib graph, then continually adapts on Blueprint dependencies.

Experiments show that **DAPS** outperforms strong prior selectors across all four benchmarks with varying distributions. The lead persists at small retrieval budgets on **DAPS**-Blueprint. Substituting **DAPS** premises for those from existing Lean search engines improves the performance of general-purpose LLMs in informal theorem proving.

Our contributions are threefold:

- We diagnose three intrinsic limitations of existing premise selectors: long-tail collapse (P1), group-level relevance (P2), and distribution shift (P3).
- We bring underused Lean 4 dependency resources to bear on premise selection, specifically by systematically mining Lean Blueprint projects and comprehensive Mathlib4 dependencies. Derived from them, we release four evaluation benchmarks of increasing dis-

[3] https://github.com/PatrickMassot/leanblueprint



(a) Typed edge spectrum.

(b) In-/out-degree asymmetry.

Figure 2. Mathlib4's dependency graph carries a rich, asymmetric, frequency-skewed structure. (a) 15 typed edge categories totaling 6.5M edges. (b) Average in- vs. out-degree per node category (bubble area $\propto \log_{10}$ #nodes). Statements (red) sit at the high-out/low-in extreme, foundational type-defining constructs (blue) at the opposite extreme.

tribution shift: Mathlib4-Random, Mathlib4-Heldout, **DAPS**-Collection, and **DAPS**-Blueprint.

- We introduce **DAPS**, a dependency-aware premise selector that outperforms strong prior selectors, holds its lead at small retrieval budgets on **DAPS**-Blueprint, and improves downstream informal theorem proving.

2. Related Works

Premise selection. Hammers (Czajka & Kaliszyk, 2018; Paulson, 2010; Zhu et al., 2025), as a significant line of work, primarily rely on brute-force proof state suggestions, which are not well supported by LLM provers. Some advanced works (Wu, 2022; Jiang et al., 2022; First et al., 2023; Mikula et al., 2024) develop frameworks for leveraging the advantages of both hammers and LLMs, where the LLMs are trained for tool use instead of using the premises. On the other hand, decomposition-based methods (Jiang et al., 2023; Wang et al., 2024; Cao et al., 2025) solely break down proof goals into simpler objectives or tractable subgoals, using self-generated premises. Retrieval-augmented provers (Yang et al., 2023; Shen et al., 2025) passively receive retrieved premises solely relying on semantic similarity. **DAPS** addresses this bottleneck by augmenting each candidate's text representation with its position in the Lean dependency graph and training on premise groups rather than individual lemmas, **DAPS** produces top-K premise sets whose collective utility is more legible to theorem proving.

Graph- and dependency-aware premise selection. A separate line of work treats premise selection as graph-structured retrieval over formula syntax or proof dependencies. Building on deep graph embeddings of formula trees (Wang et al., 2017), subsequent work strengthens the structural component via subgraph pooling (Crouse et al., 2019), higher-order logic encodings (Paliwal et al., 2020), graph contrastive pretraining (Li et al., 2021), and graph sequence learning (Holden & Korovin, 2025). Existing Lean dependency extractors are limited in different respects: lean-graph (Cihal, 2024) resolves precise fully-qualified names through Lean meta-programming but covers only four node types and depth-bounded subgraphs from a root, import-graph (Community, 2024) captures only module-level import edges and requires a successful `lake build`, and the ntp-toolkit underlying LeanHammer (Zhu et al., 2025) recovers tactic-implicit (`rw`, `simp`) references but restricts nodes to theorems and definitions. **DAPS** contributes a richer extraction pipeline with the broadest node and edge schema, no Lean environment or successful build required, scales to the full Mathlib4 library with node-level file reconstruction, and extends to a previously untapped Blueprint corpus. It trains on this graph with a structural neighborhood encoder and group-level contrastive objective, rather than the per-formula syntax graphs and single-positive similarity ranking used by prior graph-based selectors.

LLM theorem proving. LLM-based theorem proving has evolved from transformer-based step generation (Polu & Sutskever, 2020) and LLM-driven goal decomposition (Jiang et al., 2023; Wang et al., 2024) to large-scale RL-trained whole-proof models such as DeepSeek-Provers (Xin et al., 2024; 2025; Ren et al., 2025), Kimina-Prover (Wang et al., 2025), and Goedel-Provers (Lin et al., 2025a;b), alongside natural-language provers such as Gemini Deep Think (Luong et al., 2025b) and Aletheia (Feng et al., 2026). All operate self-contained, relying on internally generated

subgoals and memorized premises, overlooking the possibility of using newly established and rare theorems from external databases. **DAPS** provides the premise-side complement by surfacing rare and recently introduced lemmas from Lean libraries, enabling general-purpose LLMs to prove theorems beyond their training distribution.

3. Dependency Resources

We assemble two Lean 4 dependency resources, the typed dependency graph of Mathlib4 (Section 3.1) and a mined corpus of 53 open-source Lean Blueprint projects (Section 3.2), and derive four evaluation benchmarks from them (Section 3.3).

3.1. Mathlib Graph

We compile all Lean 4 source files in `mathlib-4.22.0`, which yields the corresponding compiled constants in the `*.olean` files for each formal declaration, including `theorems`, `lemmas`, `definitions`, `instances`, and so forth. Such compiled constants represent the dependencies. For each formal declaration, we also record its `type` and `docstring`. We disable notation pretty printing and extend the constants with their fully qualified name, following LeanHammer (Zhu et al., 2025). Because the goal of training **DAPS** is to bring together premises applicable for proving a specific theorem, for each declaration, we further parse its statement and the whole proof from the source Lean file. We record the line number and namespace for precise positioning and reconstruction. Additionally, we record the containment relationship between each declaration and its file, as well as the import relationship between files. As a result, these dependencies form a multi-level directed acyclic graph, where each node is either a formal declaration or a file, and each edge represents a typed relationship. Figure 2a shows the edge spectrum of the constructed dependency graph. Appendix E.1 shows the complete node and edge categories, and further analysis.

The graph also carries strong directional structure (Figure 2b). Statement categories `theorem` and `lemma`, which jointly comprise 56% of all declarations, have a low average in-degree (3–5) but a high average out-degree (27–31). These categories consume many premises but are rarely premises themselves. Foundational categories such as `class`, `structure`, and `instance` are opposite, with average in-degree up to two orders of magnitude higher and out-degree below 6. Premise selection is therefore inherently asymmetric retrieval, in which queries in the high-out, low-in end must be matched against premises in the high-in, low-out end. We encode in- and out-degree as separate features in **DAPS** to capture this asymmetry.

3.2. Blueprint Mining

In Lean 4, Blueprint[4] is a tool to manage collaborative formalization projects. For each project, a dependency graph is automatically generated from the human-written chapters of cross-referenced definitions, lemmas, propositions, theorems, and corollaries, which are initially written in LaTeX and then progressively manually formalized in Lean 4. While some of these mathematical statements have been integrated into Mathlib, others have been written anew by mathematicians, and the dependencies are recorded solely in the blueprint. This is especially the case for advanced theorem formalization projects. As a result, the blueprint dependency graphs contain rich information that is not presented in the Mathlib: (1) Each node has a high-quality informal statement and proof; (2) The graph contains infrequent nodes out of Mathlib. These provide high-quality and infrequent data for training **DAPS**.

We collect blueprints from open-sourced Blueprint projects. We exclude low-quality items, including those with an excessively high proportion of missing formalized statements and those with a limited number of dependencies. This results in 53 blueprint dependency graphs with a total of 3,805 nodes. Table 12 in Appendix E.2 includes detailed statistics.

Table 1 summarizes the resulting corpus, and Figure 3 breaks it down by project. The corpus contains 3,805 nodes and 4,651 dependency edges, split 71.7%/28.3% between formal and informal nodes. Project sizes are long-tailed: the median project contains 35 nodes, the mean 72, and a handful of research-level efforts account for over half of the total, while the remaining 38 smaller projects form the low-frequency tail. Such research-level projects include `PrimeNumberTheoremAnd` (525 nodes), `brownian-motion` (458), `pfr` (218), `equational_theories` (214), and the eleven FLT sub-chapters (239 combined). Together, these projects cover number theory, analysis, algebra, topology, and combinatorics, with no single area dominating.

Crucially, the formal/informal proportion varies widely across projects. Per-project formal ratio ranges from 29% (`FormalBook`) to 100% (`sphere-eversion`, `liquid-s1`), reflecting the full life-cycle from early-stage research blueprint to completed formalization. 14 are fully formalized, 14 remain informal-majority, and the rest sit in a mixed state. This is precisely what makes Blueprints a useful complement to Mathlib4, where every project contributes both fully-checked formal anchor nodes and informal frontier nodes that have yet to be formalized. The latter are the rare, high-value premises that no purely Mathlib-trained selector ever sees.

[4] https://github.com/PatrickMassot/leanblueprint

Table 1. Mined Lean Blueprint corpus (full per-project listing in Appendix E.2).

# Projects	53
# Nodes / # Edges	3,805 / 4,651
Formal vs. informal nodes	2,728 (71.7%) / 1,077 (28.3%)
Project size: median / mean / range	35 / 71.8 / 6–525
Fully-formalized projects	14
Informal-majority projects	14

Table 2. Training and evaluation data.

Source	**#Train**	**#Val**	**Test slices (#queries)**
Mathlib4	275,842	2,000	Mathlib4-Heldout (1,000), Mathlib4-Random (1,000)
Blueprints	1,880	235	**DAPS**-Blueprint (236)
DAPS-Collection	–	–	**DAPS**-Collection (1,000)

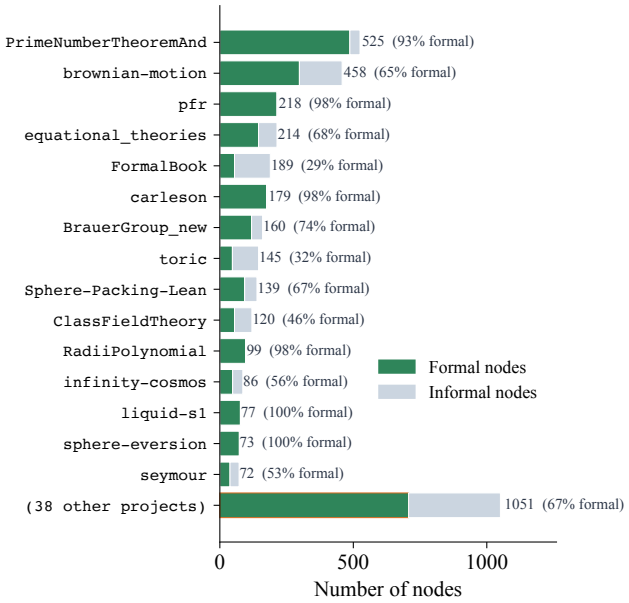


Figure 3. Composition of the mined Blueprint corpus. The top-15 projects are shown individually, while the remaining 38 are aggregated into the bottom bar. Bars are divided into formal nodes (green; statements with Lean proofs) and informal nodes (gray; LaTeX-only statements, sometimes with informal proofs).

3.3. Data Splits and Benchmarks

We use two training sources and one test-only out-of-distribution source (Table 2). **Mathlib4-Random** (1,000 queries) samples queries at random across all files, disjoint from the training and validation sets. **Mathlib4-Heldout** (1,000 queries) holds out entire source files from training, a more conservative split than the theorem-level holdout used by prior work (Zhu et al., 2025). **DAPS-Collection** (1,000 queries) probes query-side distribution shift. Queries are sampled from 152,038 `(problem, proof)` pairs aggregated from three open-source Lean 4 datasets in Olympiad and synthetic styles, with candidates still drawn from Mathlib4. **DAPS-Blueprint** (236 queries) is the test slice of an 8:1:1 split of our Blueprint corpus, stressing rare premises and informal queries. Full split methodology and source datasets are in Appendix E.3.

4. The DAPS Selector

DAPS is introduced to address the sketched problems Section 1 with: a structural neighborhood encoder for P1 (Section 4.2), a group-level training objective for P2 (Section 4.3), and a two-stage adaptation procedure for P3 (Section 4.4). We first formalize the task.

4.1. Problem Formulation

Let $\mathcal{L} = (\mathcal{V}, \mathcal{E})$ denote a Lean library represented as a typed multi-level dependency DAG, where $\mathcal{V}$ is the set of declarations and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{T}$ is a typed edge set with edge categories $\mathcal{T}$ (Section 3.1). For each query declaration $q \in \mathcal{V}$, let $\mathcal{P}^\star(q) \subseteq \mathcal{V}$ denote its *positive* premises — the declarations the verified proof of q depends on. Given a candidate pool $\mathcal{C}(q)$, premise selection produces a ranking $r : \mathcal{C}(q) \to \mathbb{R}$ that places $\mathcal{P}^\star(q)$ at the top, evaluated by Recall@K:

$$\mathrm{R@}K(q) \;=\; \frac{|\,\mathrm{Top}_K(r) \cap \mathcal{P}^\star(q)\,|}{|\mathcal{P}^\star(q)|}.$$

DAPS instantiates r by learning dependency-based embeddings $h(\cdot)$ of queries and candidates and ranking by cosine similarity $\mathrm{sim}(h(q), h(p))$.

4.2. Structural Neighborhood Encoding

For each candidate $p \in \mathcal{C}(q)$, we encode both p itself and a sampled neighborhood from its 2-hop dependency subgraph in $\mathcal{L}$. The neighborhood signal is what allows **DAPS** to rank rare premises whose own declaration text is short or unfamiliar (P1).

Semantic encoding. A small text encoder maps the concatenation of `name`, `category`, `signature`, `docstring`, `statement`, and `proof` of any declaration p to a base feature $h_{\mathrm{feat}}(p) = \mathrm{Encoder}(p)$. The encoder is fine-tuned with the masked contrastive loss of (Zhu et al., 2025), with queries and premises encoded independently and their embeddings contrasted on (q, p_{pos}) and (q, p_{neg}) pairs, and then frozen for the rest of **DAPS** training (Appendix F).

Dynamic subgraph sampling. For each candidate p, we collect the 1- and 2-hop neighbors of p in $\mathcal{L}$, randomly sample m of them, score the m candidates with a learnable two-layer FFN over $h_{\text{feat}}(\cdot)$, and retain the top-n scored neighbors as $\mathcal{N}(p)$. This sample-then-score selection bounds compute and memory while preferentially retaining structurally informative neighbors. It also matches the deployment-time constraint that downstream LLM provers consume only a bounded premise list.

Per-neighbor structural features. We adapt three token features from the relational graph transformer (RelGT) (Dwivedi et al., 2025) and add one feature specific to typed dependency graphs. For each $p_i \in \{p\} \cup \mathcal{N}(p)$ we compute:

1. *Type* embedding $h_{\text{type}}(p_i)$ via a one-hot over the `subcategory` field (Dwivedi et al., 2025);
2. *Hop* embedding $h_{\text{hop}}(p, p_i) \in \{0, 1, 2\}$ giving the dependency hop distance from p to p_i (Dwivedi et al., 2025);
3. *Positional* encoding $h_{\text{pe}}(p_i)$ over the sampled local subgraph (Dwivedi et al., 2025);
4. *Degree* embedding $h_{\text{deg}}(p_i) = W_{\text{deg}}\,[\deg^-(p_i); \deg^+(p_i)]$ of in- and out-degree, which is motivated by the directional asymmetry of the dependency graph (Section 3.1, Figure 2b), in which different declaration categories occupy distinct in- vs. out-degree regions.

Fusion. Per-neighbor representations are merged by a two-layer MLP O, with the candidate's own type re-injected through a residual path so it cannot be lost in subgraph aggregation:

$$h(p_i) = O\left[h_{\text{feat}}(p_i);\, h_{\text{type}}(p_i);\, h_{\text{hop}}(p, p_i);\, h_{\text{pe}}(p_i)\right] + h_{\text{deg}}(p_i) + h_{\text{res}}(p), \tag{1}$$

where $h_{\text{res}}(p) = W_{\text{type}}\,\text{onehot}(\phi(p))$.

4.3. Group-Level Contrastive Objective

A useful proof typically requires several premises working together. Therefore, unlike the typical contrastive setup that a query is pulled toward a single positive sample, **DAPS** is designed to rank a large set of candidate premises, in which multiple positives are pulled toward the query *as a group* relative to the negatives (P2).

For each query q in a training batch, we sample a positive set $\mathcal{P}_q^+ \subseteq \mathcal{P}^\star(q)$ (the top-ℓ positives under the category priority of Appendix F) and a negative set $\mathcal{P}_q^-$ via three complementary strategies, drawing on standard hard-negative mining techniques: (i) *temporal* sampling of k_t declarations preceding q in the same or imported files; (ii) *global* sampling of k_g declarations uniformly from $\mathcal{L}$; and (iii) *cross-batch* sampling of k_c candidates re-used from other queries in the batch, after masking those that collide with $\mathcal{P}^\star(q)$. The loss is

$$\mathcal{L}_q = -\frac{1}{|\mathcal{P}_q^+|} \sum_{p \in \mathcal{P}_q^+} \log \frac{\exp\big(\text{sim}(h(q),\, h(p))/\tau\big)}{\sum_{p' \in \mathcal{P}_q^+ \cup \mathcal{P}_q^-} \exp\big(\text{sim}(h(q),\, h(p'))/\tau\big)},$$

with temperature τ. Each positive contributes its own log-term but shares the denominator with the rest of $\mathcal{P}_q^+$ and all negatives, encouraging the entire positive group to dominate the ranking rather than only the closest one.

4.4. Mathlib-to-Blueprint Adaptation

Lean usage spans diverse distributions (P3). To transfer the structural signal learned on Mathlib4 to the long tail of advanced and not-yet-formalized premises in the Blueprint corpus (Section 3.2), **DAPS** uses a two-stage training procedure. *(1) Mathlib4 pretraining* yields strong representations of typed dependency graph structure on the abundant Mathlib data. *(2) Blueprint continual training* then fine-tunes the structural fusion modules on the Blueprint training split, with a smaller dynamic-sampling budget reflecting the smaller corpus size. The queries are informal statements, and the candidate premises are encoded with both their formal and informal statements. The implementation details are in Appendix F We report both the Mathlib-only and the continually-trained model in Section 5, demonstrating that the typed-graph signal learned on Mathlib transfers, and that Blueprint adaptation further improves recall on rare and advanced premises.

4.5. Informal-Query Adaptation

When the query itself is informal (i.e., a natural-language theorem statement), **DAPS** is paired with a formalizer $\mathcal{F}$ that produces a Lean statement $\tilde{q} = \mathcal{F}(q_{\text{nl}})$. **DAPS** then operates on $\tilde{q}$ as a formal query and returns Lean-side premises. This formalize-then-retrieve composition lets downstream systems that consume informal proofs, such as general-purpose LLMs, still benefit from dependency-aware retrieval, without **DAPS** itself ever processing natural language.

5. Experiments

5.1. Experimental Setup

Datasets. We evaluate on the four benchmarks introduced in Section 3.3: Mathlib4-Random and Mathlib4-Heldout for in-distribution and within-Mathlib generalization, **DAPS**-Collection for query-side distribution shift to Olympiad and synthetic problems, and **DAPS**-Blueprint for retrieval

Table 3. Recall@32 across all four benchmarks. For fair comparison, ReProver and LeanHammer are retrained on the same Mathlib dependency data using their published recipes. MePo (symbolic) and Qwen3-Embedding-8B (pretrained encoder) are used as released. †: Mathlib-only model. ‡: After Blueprint continual training. Mathlib4-R: Mathlib4-Random. Mathlib4-H: Mathlib4-Heldout.

Method	**Mathlib4-R**	**Mathlib4-H**	**DAPS-Collection**	**DAPS-Blueprint**
MePo (Meng et al., 2006)	19.36	11.94	1.08	–
ReProver (Yang et al., 2023)	36.97	43.48	1.24	32.04
LeanHammer (Zhu et al., 2025)	73.99	77.84	3.19	40.42
Qwen3-Embedding-8B (Zhang et al., 2025b)	15.25	9.92	3.59	11.31
DAPS	**79.63** †	**89.31** †	**24.38** †	**49.90** ‡

Table 4. Downstream utility performance. Pass@8 of different LLMs on IMO-ProofBench (Luong et al., 2025a) using premises from compared premise selectors. In this table, we strictly follow the original grading prompts from the IMO-ProofBench paper and use the identical grading model.

Model	**LeanExplore**	**LeanSearch**	**Qwen3-Embedding-8B**	**DAPS**
Qwen3-235b (Yang et al., 2025)	18.45	17.01	19.08	**21.25**
DeepSeek-V3.1 (DeepSeek-AI, 2024)	11.96	12.38	13.75	**13.90**
kimi-k2-instruct (Bai et al., 2025)	8.66	8.26	8.27	**9.08**

against the rare, advanced, and informal+formal premises. Split sizes follow Table 2.

Baselines. We group baselines by the experiment in which each is evaluated.

Retrieval baselines: the symbolic selector **MePo** (Meng & Paulson, 2009), the LeanDojo-trained **ReProver** (Yang et al., 2023) based on the small variant of Google ByT5 (Xue et al., 2022), the sentence-transformer-based **LeanHammer** (Zhu et al., 2025), and the general-purpose **Qwen3-Embedding-8B** (Zhang et al., 2025b). To ensure a fair comparison, we retrain ReProver, LeanHammer, and Qwen3-Embedding-8B on our Mathlib dependency data using their published recipes. MePo is symbolic and used as released. We follow LeanHammer in calling the MePo premise selector implemented in Lean[5].

Downstream-utility baselines: in addition to Qwen3-Embedding-8B above, we compare against two production-grade Lean search engines that natively accept informal natural-language queries: **LeanExplore** (Asher, 2025) and **LeanSearch** (Gao et al., 2024). Both are used as released APIs.

Metrics. We report Recall@K for retrieval, with $K \in \{8, 16, 32, 64\}$; Recall@32 serves as the headline metric. For the informal proving experiments, we follow the IMO-ProofBench protocol (Luong et al., 2025a) and report grading scores from the IMO-ProofBench autograder, using the same grading prompts and grading model as the original benchmark.

[5] https://github.com/hanwenzhu/premise-selection

Infrastructure. Training and inference compute, as well as hardware specifications, are reported in Appendix D.

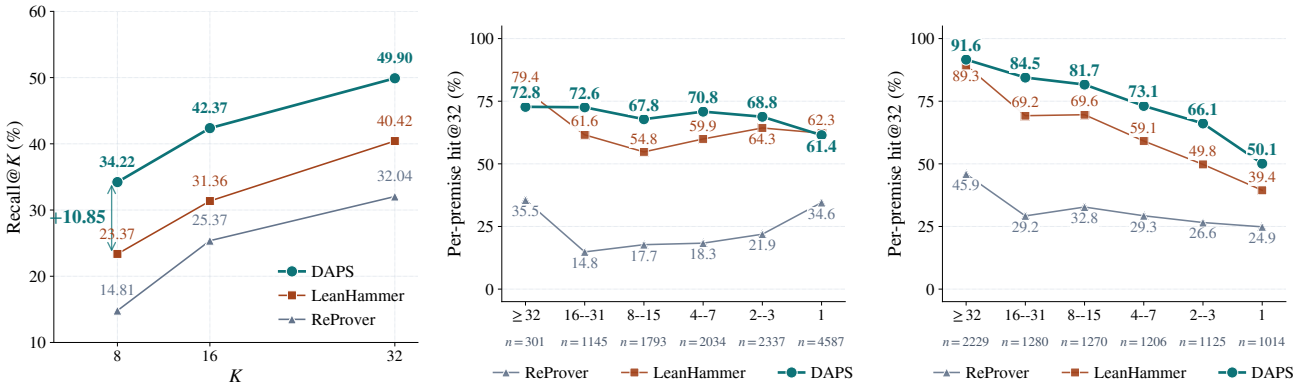
5.2. Main Results

Table 3 reports Recall@32 across the four benchmarks. Three observations stand out. (1) For in-distribution Mathlib retrieval (Mathlib4-Random/Mathlib4-Heldout), **DAPS** reaches 79.63/89.31, an absolute gain of +5.64/+11.47 over the strong prior selector LeanHammer (73.99/77.84). (2) For out-of-distribution shift to competition-style queries (**DAPS**-Collection), all prior selectors drop to single-digit recall, with the best baseline of 3.59. **DAPS** reaches 24.38, a 7× improvement that demonstrates the value of dependency-graph signal under query distribution shift. (3) For rare, advanced, and informal+formal premises (**DAPS**-Blueprint), **DAPS** with Blueprint continual training reaches 49.90, +9.48 over LeanHammer (40.42).

Figure 4a reports Recall@K on **DAPS**-Blueprint across $K \in \{8, 16, 32\}$, extending the K=32 comparison of Table 3. **DAPS** dominates at every K, with absolute gains of +10.85, +11.01, and +9.48 over LeanHammer at $K = 8$, 16, and 32, respectively. The gap remains large even at small K, demonstrating **DAPS**'s value for downstream LLMs with limited context windows for premises.

5.3. Adaptation to Informal Theorem Proving

We test whether **DAPS**'s retrieval improvements transfer to a downstream proving task. We focus on the *informal-proving* setting in which a general-purpose LLM is asked to produce a natural-language proof, optionally conditioned on a list of retrieved Lean premises.



(a) Recall@K on **DAPS**-Blueprint.

(b) Premise-frequency breakdown on Mathlib4-Random.

(c) Premise-frequency breakdown on Mathlib4-Heldout.

Figure 4. (a) Recall@K on **DAPS**-Blueprint for $K \in \{8, 16, 32\}$. (b) Per-pair hit@32 by premise frequency on Mathlib4-Random. (c) Per-pair hit@32 by premise frequency on Mathlib4-Heldout.

Setup. We follow the adaptation of Section 4.5: an informal IMO-ProofBench (Luong et al., 2025a) query is first formalized into a Lean statement using Gemini 3 Pro[6], the resulting Lean statement is fed to each selector to obtain the top-16 Mathlib premises, and the original informal query together with the selected premises is given to a general-purpose LLM that produces an informal proof. The proof is graded by the IMO-ProofBench autograder using the original grading prompt.

Results. Table 4 reports IMO-ProofBench scores across three frontier general-purpose LLMs (Qwen3-235B (Yang et al., 2025), DeepSeek-V3.1 (DeepSeek-AI, 2024), Kimi-K2-Instruct (Bai et al., 2025)) under four selector settings (LeanExplore (Asher, 2025), LeanSearch (Gao et al., 2024), Qwen3-Embedding-8B (Zhang et al., 2025b), and **DAPS**). For a fair comparison, each selector receives the same formalized query and is constrained to return the same number of top-K premises, where $K = 16$. We report Pass@8 only, owing to the cost of the IMO-ProofBench autograder. Across all three LLMs, premises returned by **DAPS** yield the highest grades, demonstrating that dependency-aware selection on the formal side translates to better informal proving on the LLM informal theorem proving side. The absolute gains over Qwen3-Embedding-8B range from +0.15 to +2.17 absolute points. Over the LLMs, the gains are largest for Qwen3-235B, consistent with the hypothesis that more capable models extract more signal from higher-quality retrieved context.

5.4. Analysis

Ablation study on training objective. To isolate the contribution of the group-level contrastive objective, we replace it with single-positive InfoNCE (Oord et al., 2018). This drops Recall@32 to 56.48 on Mathlib4-Random, 56.00 on Mathlib4-Heldout, and 15.25 on **DAPS**-Collection, an absolute decrease of 23.14, 33.30, and 9.13 from the full **DAPS** configuration. The result confirms the value of group-level relevance as a training signal. Further ablation studies are deferred to Appendix G.1.

Premise frequency breakdown. On Mathlib4-Random and Mathlib4-Heldout, we collect all `(query` q`, gold premise` p`)` pairs from the test set and bucket each premise by its test-set frequency, i.e., the number of queries that list it as gold. We report mean per-pair hit@32 within each bucket, which indicates p appears in the selector's top-32 retrievals for q, averaged within the bucket. On Mathlib4-Random, **DAPS** and LeanHammer achieve broadly comparable per-bucket hit rates while ReProver trails consistently. LeanHammer narrowly leads on the head bucket and the singletons ($n = 1$), while **DAPS** leads in the four middle buckets. Semantic similarity is therefore sufficient when the training and test premise distributions are closely matched. The picture changes on Mathlib4-Heldout, whose file-level isolation introduces a moderate distribution shift on the query side. **DAPS** leads LeanHammer in every bucket, by 2.3 points on the head (91.6 vs 89.3) and 10.7 on the rarest bucket (50.1 vs 39.4), with the gap widening toward the tail. This matches the P1 prediction that rare premises carry less text signal, so baselines miss them more often once the test distribution shifts. **DAPS** lifts performance uniformly across the frequency spectrum.

Case studies and limitations. We provide additional qualitative case studies and a failure-mode analysis for **DAPS** in Appendices G.2 and G.3. Detailed limitations are in Appendix A.

[6] https://deepmind.google/models/gemini/pro/

6. Conclusion

We present three intrinsic limitations of current premise selection methods: frequency-skewed retrieval (P1), inability to model group-level relevance (P2), and brittleness under distribution shift (P3), all tracing to the underexploitation of the built-in mathematics dependency structure. We curate two underused Lean 4 dependency resources, a typed multi-level graph extracted from Mathlib4 and systematic mining of Lean Blueprint projects, and derive four retrieval benchmarks of increasing distribution shift. We introduce **DAPS**, a dependency-aware selector with a structural neighborhood encoder, a group-level contrastive objective, and a Mathlib-then-Blueprint adaptation procedure that targets P1–P3, respectively. **DAPS** outperforms strong prior selectors across four benchmarks (+11.47 Recall@32 over LeanHammer on Mathlib4-Heldout), holds its lead at small retrieval budgets on **DAPS**-Blueprint, and improves general-purpose LLMs on informal IMO-ProofBench.

References

Asher, J. Leanexplore: A search engine for lean 4 declarations. *CoRR*, abs/2506.11085, 2025. doi: 10.48550/ARXIV.2506.11085. URL https://doi.org/10.48550/arXiv.2506.11085.

Bai, Y., Bao, Y., Chen, G., Chen, J., Chen, N., Chen, R., Chen, Y., Chen, Y., Chen, Y., Chen, Z., Cui, J., Ding, H., Dong, M., Du, A., Du, C., Du, D., Du, Y., Fan, Y., Feng, Y., Fu, K., Gao, B., Gao, H., Gao, P., Gao, T., Gu, X., Guan, L., Guo, H., Guo, J., Hu, H., Hao, X., He, T., He, W., He, W., Hong, C., Hu, Y., Hu, Z., Huang, W., Huang, Z., Huang, Z., Jiang, T., Jiang, Z., Jin, X., Kang, Y., Lai, G., Li, C., Li, F., Li, H., Li, M., Li, W., Li, Y., Li, Y., Li, Z., Li, Z., Lin, H., Lin, X., Lin, Z., Liu, C., Liu, C., Liu, H., Liu, J., Liu, J., Liu, L., Liu, S., Liu, T. Y., Liu, T., Liu, W., Liu, Y., Liu, Y., Liu, Y., Liu, Y., Liu, Z., Lu, E., Lu, L., Ma, S., Ma, X., Ma, Y., Mao, S., Mei, J., Men, X., Miao, Y., Pan, S., Peng, Y., Qin, R., Qu, B., Shang, Z., Shi, L., Shi, S., Song, F., Su, J., Su, Z., Sun, X., Sung, F., Tang, H., Tao, J., Teng, Q., Wang, C., Wang, D., Wang, F., and Wang, H. Kimi K2: open agentic intelligence. *CoRR*, abs/2507.20534, 2025. doi: 10.48550/ARXIV.2507.20534. URL https://doi.org/10.48550/arXiv.2507.20534.

Cao, C., Song, L., Li, Z., Le, X., Zhang, X., XUE, H., and Yang, F. Reviving DSP for advanced theorem proving in the era of reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=yTFJmGFsEy.

Cihal, P. lean-graph: Visualizing lean dependency graphs. https://github.com/patrik-cihal/lean-graph, 2024. Accessed: 2026.

Community, L. P. import-graph: Tools for working with module dependencies in lean projects. https://github.com/leanprover-community/import-graph, 2024. Accessed: 2026.

Crouse, M., Abdelaziz, I., Cornelio, C., Thost, V., Wu, L., Forbus, K. D., and Fokoue, A. Improving graph neural network representations of logical formulae with subgraph pooling. *CoRR*, abs/1911.06904, 2019. URL http://arxiv.org/abs/1911.06904.

Czajka, Ł. and Kaliszyk, C. Hammer for coq: Automation for dependent type theory. *Journal of automated reasoning*, 61(1):423–453, 2018.

DeepSeek-AI. Deepseek-v3 technical report. *CoRR*, abs/2412.19437, 2024. doi: 10.48550/ARXIV.2412.19437. URL https://doi.org/10.48550/arXiv.2412.19437.

Dwivedi, V. P., Jaladi, S., Shen, Y., López, F., Kanatsoulis, C. I., Puri, R., Fey, M., and Leskovec, J. Relational graph transformer. *CoRR*, abs/2505.10960, 2025. doi: 10.48550/ARXIV.2505.10960. URL https://doi.org/10.48550/arXiv.2505.10960.

Feng, T., Trinh, T. H., Bingham, G., Hwang, D., Chervonyi, Y., Jung, J., Lee, J., Pagano, C., Kim, S.-h., Pasqualotto, F., et al. Towards autonomous mathematics research. *arXiv preprint arXiv:2602.10177*, 2026.

First, E., Rabe, M. N., Ringer, T., and Brun, Y. Baldur: Whole-proof generation and repair with large language models. In Chandra, S., Blincoe, K., and Tonella, P. (eds.), *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pp. 1229–1241. ACM, 2023. doi: 10.1145/3611643.3616243. URL https://doi.org/10.1145/3611643.3616243.

Gao, G., Ju, H., Jiang, J., Qin, Z., and Dong, B. A semantic search engine for mathlib4. In Al-Onaizan, Y., Bansal, M., and Chen, Y. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pp. 8001–8013. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-EMNLP.470. URL https://doi.org/10.18653/v1/2024.findings-emnlp.470.

Holden, E. K. and Korovin, K. Graph sequence learning for premise selection. *J. Symb. Comput.*, 128:102376, 2025. doi: 10.1016/J.JSC.2024.102376. URL https://doi.org/10.1016/j.jsc.2024.102376.

Jiang, A. Q., Li, W., Tworkowski, S., Czechowski, K., Odrzygózdz, T., Milos, P., Wu, Y., and Jamnik, M. Thor: Wielding hammers to integrate language models and automated theorem provers. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/377c25312668e48f2e531e2f2c422483-Abstract-Conference.html.

Jiang, A. Q., Welleck, S., Zhou, J. P., Lacroix, T., Liu, J., Li, W., Jamnik, M., Lample, G., and Wu, Y. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

Li, Z., Chen, B., and Si, X. Graph contrastive pre-training for effective theorem reasoning. *CoRR*, abs/2108.10821, 2021. URL https://arxiv.org/abs/2108.10821.

Lin, Y., Tang, S., Lyu, B., Wu, J., Lin, H., Yang, K., LI, J., Xia, M., Chen, D., Arora, S., and Jin, C. Goedel-prover: A frontier model for open-source automated theorem proving. In *Second Conference on Language Modeling*, 2025a. URL https://openreview.net/forum?id=x2y9i2HDjD.

Lin, Y., Tang, S., Lyu, B., Yang, Z., Chung, J., Zhao, H., Jiang, L., Geng, Y., Ge, J., Sun, J., Wu, J., Gesi, J., Lu, X., Acuna, D., Yang, K., Lin, H., Choi, Y., Chen, D., Arora, S., and Jin, C. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *CoRR*, abs/2508.03613, 2025b. doi: 10.48550/ARXIV.2508.03613. URL https://doi.org/10.48550/arXiv.2508.03613.

Luong, T., Hwang, D., Nguyen, H. H., Ghiasi, G., Chervonyi, Y., Seo, I., Kim, J., Bingham, G., Lee, J., Mishra, S., Zhai, A., Hu, H., Michalewski, H., Kim, J., Ahn, J., Bae, J., Song, X., Trinh, T. H., Le, Q. V., and Jung, J. Towards robust mathematical reasoning. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35418–35442, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1794. URL https://aclanthology.org/2025.emnlp-main.1794/.

Luong, T., Lockhart, E., et al. Advanced version of gemini with deep think officially achieves gold-medal standard at the international mathematical olympiad. *Google DeepMind Blog*, 1, 2025b.

Meng, J. and Paulson, L. C. Lightweight relevance filtering for machine-generated resolution problems. *Journal of Applied Logic*, 7(1):41–57, 2009.

Meng, J., Quigley, C., and Paulson, L. C. Automation for interactive proof: First prototype. *Inf. Comput.*, 204(10): 1575–1596, 2006. doi: 10.1016/J.IC.2005.05.010. URL https://doi.org/10.1016/j.ic.2005.05.010.

Mikula, M., Tworkowski, S., Antoniak, S., Piotrowski, B., Jiang, A. Q., Zhou, J. P., Szegedy, C., Kucinski, L., Milos, P., and Wu, Y. Magnushammer: A transformer-based approach to premise selection. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=oYjPk8mqAV.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Paliwal, A., Loos, S. M., Rabe, M. N., Bansal, K., and Szegedy, C. Graph representations for higher-order logic and theorem proving. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pp. 2967–2974. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5689.

Paulson, L. C. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Schmidt, R. A., Schulz, S., and Konev, B. (eds.), *Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning, PAAR-2010, Edinburgh, Scotland, UK, July 14, 2010*, volume 9 of *EPiC Series in Computing*, pp. 1–10. EasyChair, 2010. doi: 10.29007/TNFD. URL https://doi.org/10.29007/tnfd.

Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020. URL https://arxiv.org/abs/2009.03393.

Ren, Z. Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., Wu, Z. F., Gou, Z., Ma, S., Tang, H., Liu, Y., Gao, W., Guo, D., and Ruan, C. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *CoRR*, abs/2504.21801, 2025. doi: 10.48550/ARXIV.2504.21801. URL https://doi.org/10.48550/arXiv.2504.21801.

Shen, Z., Huang, N., Yang, F., Wang, Y., Gao, G., Xu, T., Jiang, J., He, W., Yang, P., Sun, M., Ju, H., Wu, P., Dai, B., and Dong, B. Real-prover: Retrieval augmented lean prover for mathematical reasoning. *CoRR*, abs/2505.20613, 2025. doi: 10.48550/ARXIV.2505.20613. URL https://doi.org/10.48550/arXiv.2505.20613.

The mathlib Community. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, New Orleans, LA, USA, January 2020. ACM. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145/3372885.3373824.

Wang, H., Xin, H., Zheng, C., Liu, Z., Cao, Q., Huang, Y., Xiong, J., Shi, H., Xie, E., Yin, J., Li, Z., and Liang, X. Lego-prover: Neural theorem proving with growing libraries. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=3f5PALef5B.

Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., Santos, M. D., Sung, F., Vinyes, M., Ying, Z., Zhu, Z., Lu, J., de Saxcé, H., Bailey, B., Song, C., Xiao, C., Zhang, D., Zhang, E., Pu, F., Zhu, H., Liu, J., Bayer, J., Michel, J., Yu, L., Dreyfus-Schmidt, L., Tunstall, L., Pagani, L., Machado, M., Bourigault, P., Wang, R., Polu, S., Barroyer, T., Li, W., Niu, Y., Fleureau, Y., Hu, Y., Yu, Z., Wang, Z., Yang, Z., Liu, Z., and Li, J. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *CoRR*, abs/2504.11354, 2025. doi: 10.48550/ARXIV.2504.11354. URL https://doi.org/10.48550/arXiv.2504.11354.

Wang, M., Tang, Y., Wang, J., and Deng, J. Premise selection for theorem proving by deep graph embedding. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 2786–2796, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/18d10dc6e666eab6de9215ae5b3d54df-Abstract.html.

Wu, Y. Formal premise selection with language models. In *Conference on Artificial Intelligence and Theorem Proving (AITP)*, volume 4, 2022.

Xin, H., Guo, D., Shao, Z., Ren, Z., Zhu, Q., Liu, B., Ruan, C., Li, W., and Liang, X. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *CoRR*, abs/2405.14333, 2024. doi: 10.48550/ARXIV.2405.14333. URL https://doi.org/10.48550/arXiv.2405.14333.

Xin, H., Ren, Z. Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., Gao, W., Zhang, H., Zhu, Q., Yang, D., Gou, Z., Wu, Z. F., Luo, F., and Ruan, C. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=I4YAIwrsXa.

Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., and Raffel, C. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R. J., and Anandkumar, A. Leandojo: Theorem proving with retrieval-augmented language models. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html.

Ying, H., Wu, Z., Geng, Y., Wang, J., Lin, D., and Chen, K. Lean workbook: A large-scale lean problem set formalized from natural language math problems. In Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J. M., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/bf236666a2cc5f3

ae05d2e08485efc4c-Abstract-Datasets_and_Benchmarks_Track.html.

Zhang, J., Wang, Q., Ji, X., Liu, Y., Yue, Y., Zhang, F., Zhang, D., Zhou, G., and Gai, K. Leanabell-prover: Posttraining scaling in formal reasoning. *CoRR*, abs/2504.06122, 2025a. doi: 10.48550/ARXIV.2504.06122. URL https://doi.org/10.48550/arXiv.2504.06122.

Zhang, X., Zhang, Y., Long, D., Xie, W., Dai, Z., Tang, J., Lin, H., Yang, B., Xie, P., Huang, F., et al. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 1393–1412, 2024.

Zhang, Y., Li, M., Long, D., Zhang, X., Lin, H., Yang, B., Xie, P., Yang, A., Liu, D., Lin, J., Huang, F., and Zhou, J. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b.

Zhu, T., Clune, J., Avigad, J., Jiang, A. Q., and Welleck, S. Premise selection for a lean hammer. *CoRR*, abs/2506.07477, 2025. doi: 10.48550/ARXIV.2506.07477. URL https://doi.org/10.48550/arXiv.2506.07477.

A. Limitations

This work has several limitations. We treat compiled dependencies as ground truth, even though many proofs admit multiple valid paths, and Recall@K does not credit premises that would support an alternative proof. Our pipeline is also Lean 4 specific, and porting to Coq or Isabelle would require reimplementation. Finally, our downstream evaluation is restricted to informal theorem proving on IMO-ProofBench, leaving end-to-end integration with formal Lean provers and prover-side feedback in the contrastive loss as natural extensions.

B. Broader Impacts

This work bridges formal and informal mathematical reasoning by mining Lean Blueprints (research-level LaTeX repositories with cross-references to in-progress Lean formalizations) and using the resulting dependency signal to improve general-purpose LLMs on informal theorem proving (Section 5.3). Positive impacts include narrowing the gap between informal research mathematics and its formal counterpart, and supporting LLM-based mathematical assistants for working mathematicians. On the negative side, DAPS reflects the coverage and biases of Mathlib4 and the Blueprint projects we mined. Users working in mathematical areas underrepresented in these libraries may see reduced retrieval quality. The released artifacts contain only formal and informal mathematical content from public open-source projects, and we do not foresee misuse risks beyond those general to AI systems for mathematical reasoning.

C. Reproducibility

We release the following at the anonymous URL[7] (also presented in the abstract) for review, under a permissive license upon publication:

- A demo of the Mathlib4 typed dependency graph and our extraction pipeline.
- A demo of the mined Blueprint corpus.
- Demos of the four evaluation benchmarks (Mathlib4-Random, Mathlib4-Heldout, **DAPS**-Collection, **DAPS**-Blueprint).
- the **DAPS** implementation, training and adaptation scripts, and evaluation pipeline.

Training hyperparameters are summarized in Table 13, with optimization details of learning-rate schedule, warmup, batching, and gradient accumulation in Appendix F. Compute requirements and runtime statistics for each training stage are reported in Appendix D.

D. Compute Reporting

Following the NeurIPS 2026 compute reporting guidance, we summarize the compute resources used to produce the experimental results in this submission.

Scope. This report covers the computation directly used for training the models reported in the main paper. The following are explicitly **not** included: (i) the foundation-model pretraining compute of third-party base models we build upon (e.g., MGTE (Zhang et al., 2024)); (ii) compute used for inference with downstream LLM provers (general-purpose LLMs in Section 5.3), which we use as off-the-shelf systems; (iii) compute consumed by preliminary experimentation, failed runs, and hyperparameter sweeps not reflected in the final reported results.

Hardware. The experimental environment is heterogeneous, comprising a GPU cluster used for the main training stages and a local workstation used for the smaller Blueprint adaptation stage.

[7] https://anonymous.4open.science/r/DAPS-61CF

- **Cluster GPUs.** NVIDIA GH200 superchips (Arm Neoverse V2 Grace CPU, 72 CPU cores per superchip; H100 GPU with 96 GB HBM3 per chip in the configuration we use). Multi-GPU jobs span up to 16 GH200 chips, corresponding to 288 CPU cores per 4-GPU node.
- **Local workstation.** 1× Apple MacBook Pro (M4 Max, unified-memory architecture).

Per-workload compute. Table 5 summarizes the compute consumed by each training workload. GPU-hours are computed as (#GPUs) × (wall-clock runtime).

Notes on the estimates. The runtimes above are wall-clock estimates from our training logs and include data-loading, checkpointing, and validation, in addition to optimizer steps. They do not include cluster idle/queue time. The local Blueprint pipeline takes less than 3 hours end-to-end, of which roughly 2 hours is the training loop itself. The remainder is data preprocessing. All reported hour counts are conservative rounded-up estimates.

E. Data Schema

E.1. Mathlib Graph

This appendix documents the complete schema of the Mathlib4 dependency graph extracted in Section 3.1. Tables 6 and 7 report the per-category counts of all nodes and typed edges in the graph. Tables 8 and 9 describe the metadata fields recorded for each node and edge, respectively. Tables 10 and 11 report the in-degree and out-degree distributions broken down by node category.

Edge typing. The top 15 typed categories together span 6.5M edges and are dominated by typeclass and instance relations, with `USES_CLASS_DERIVED` and `USES_INSTANCE` alone accounting for 1.7M and 1.3M edges, respectively, while proof-level `DEPENDS_ON` edges contribute only 618K, less than 10% of the total. Recent neural premise selectors (Yang et al., 2023; Zhu et al., 2025) typically use these dependency relations as an untyped list of used constants, without distinguishing the dependency categories above. **DAPS** instead retains edge types as first-class features (Section 4).

Frequency-skewed concentration. The asymmetry yields a heavily head-concentrated in-edge distribution (Figure 5): the most-cited 1% of nodes attract 13% of all incoming dependencies, the top 5% attract 39%, and the top 10% attract 56%. The complement is a low-frequency of theorems and lemmas with very few citations and a correspondingly weak text signal. This indicates the significance of structural features for accurate retrieval.

E.2. Blueprint Mining

This appendix lists per-project statistics of all 53 mined Lean Blueprint dependency graphs (Section 3.2), sorted by total node count. Each row corresponds to one blueprint dependency graph. Project names link to the original blueprint document. The corpus-level summary is reported in Table 1 of Section 3.2, and the corresponding visualization in Figure 3.

E.3. Full Data Splits Method and Evaluation Benchmarks

Mathlib4 splits. The Mathlib dependency graph is divided into 275,842 nodes for training, 2,000 for validation, and 2,000 for testing. The 2,000 test nodes consist of two complementary 1,000-node slices used as separate benchmarks: **Mathlib4-Heldout** is built by grouping declarations by source file, shuffling the file list, and sequentially selecting files until the cumulative declaration count first exceeds 1,000. The corresponding files are excluded from training entirely, and the first 1,000 declarations from these files form the test set. This file-level holdout is more conservative than the theorem-level random holdout used by, e.g., LeanHammer (Zhu et al., 2025), since within-file regularities (shared imports, sibling lemma context) cannot leak through training. **Mathlib4-Random** is built by randomly sampling 1,000 declarations from the remaining (non-heldout) files and excluding them from training.

DAPS-Blueprint. The Blueprint corpus is split 8:1:1 by node into train/validation/test, after dropping nodes without any recorded dependencies. The resulting test set, **DAPS-Blueprint**, contains 236 queries averaging 1.74 premises each. 26.0% are informal-only nodes (LaTeX statements without a Lean formalization).

DAPS-Collection. This dataset is held entirely out of training. We compile three open-source Lean 4 datasets under `mathlib-4.22.0`: `Goedel-LM/Lean-workbook-proofs`[8] (Ying et al., 2024; Lin et al., 2025a), `stoney0062/Leanabell-Prover-Traindata-SFT`[9] (Zhang et al., 2025a), and `AI-MO/NuminaMath-LEAN`[10], yielding 152,038 compilable `(problem, proof)` pairs in Olympiad and synthetic styles. We sample 1,000 of these and extract their dependency lists from the compiled proofs as ground-truth positives. Candidate premises still come from Mathlib, but the query distribution differs substantially from the Mathlib library, giving a

[8] https://huggingface.co/datasets/Goedel-LM/Lean-workbook-proofs

[9] https://huggingface.co/datasets/stoney0062/Leanabell-Prover-Traindata-SFT

[10] https://huggingface.co/datasets/AI-MO/NuminaMath-LEAN

Table 5. Compute consumed by the reported training workloads. Cluster runtimes are aggregated GPU-hours on NVIDIA GH200. The Blueprint adaptation stage is reported as machine-hours since it runs on a single local workstation.

Workload	Hardware	#Devices	Runtime	Compute
Stage 1: text encoder warm-up	GH200 cluster	16	$\leq$12 h	$\leq$192 GPU-h
Stage 2: structural fusion training	GH200 cluster	16	$\leq$24 h	$\leq$384 GPU-h
Blueprint continual training	MacBook Pro (M4 Max)	1	$\leq$3 h	$\leq$3 machine-h
Total (cluster training)				$\leq$408 GPU-h
Total (local training)				$\leq$3 machine-h

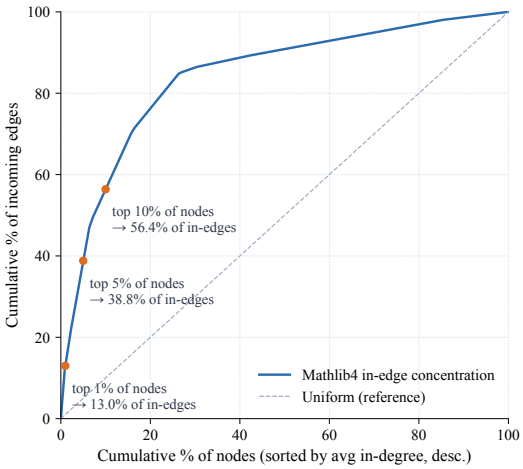


Figure 5. The top 10% of nodes attract 56% of all incoming edges (category-aggregate approximation; per-node curve is steeper).

Table 6. Statistics of node categories.

Node Category	Total
`theorem`	116,096
`lemma`	39,832
`definition`	26,540
`instance`	22,462
`definition_derived`	15,841
`theorem_derived`	14,497
`class_derived`	11,472
`structure_derived`	9,085
`file`	6,726
`lemma_derived`	3,918
`structure`	3,260
`abbrev`	2,382
`class`	2,126
`instance_derived`	1,912
`inductive_derived`	1,183
`others`	762

Table 7. Statistics of edge categories.

Edge Category	Total
`USES_CLASS_DERIVED`	1,692,183
`USES_INSTANCE`	1,285,578
`USES_DEFINITION`	838,828
`USES_CLASS`	738,150
`DEPENDS_ON`	618,222
`USES_STRUCTURE`	567,065
`CONTAINS`	273,116
`USES_INSTANCE_DERIVED`	150,501
`USES_ABBREV`	106,383
`USES_STRUCTURE_DERIVED`	77,378
`USES_DEFINITION_DERIVED`	75,626
`IMPORTS`	20,794
`USES_INDUCTIVE`	17,332
`USES_INDUCTIVE_COMPONENT_DERIVED`	15,410
`USES_INDUCTIVE_DERIVED`	3,132
`others`	3,583

clean test of generalization across query distributions while holding the candidate side fixed.

F. Implementation Details of DAPS

Mathlib4 pretraining. Queries are drawn from the `theorem` and `lemma` categories, which are the two declaration types that carry proofs and therefore yield premise lists. For each query, positive premises are taken from its compiled dependencies and prioritized by category in the order `theorem` $\succ$ `theorem_derived` $\succ$ `lemma` $\succ$ `lemma_derived` $\succ$ `definition` $\succ$ `definition_derived` $\succ$ `instance`; the top ℓ are retained. The semantic encoder reads the `statement` field for queries, and the concatenation of `name`, `category`, `signature`, `docstring`, `statement`, and `proof` for

Table 8. Description of node fields.

Node Field	Definition
name	Fully qualified Lean name (e.g., AddConstMap.instFunLike) or the synthetic file_ identifiers we create from the filesystem.
category	Underlying ConstantInfo tag (theorem, def, instance, inductive, structure, class, constructor, recursor, etc.).
line/column	Start position from findDeclarationRanges?, measured in the source .lean file.
doc	Docstring preceding the declaration; null when absent.
signature	Canonical type with all binders and constants spelled out.
isProp	Flags declarations whose type lives in Prop.
isHumanTheorem	True only for human-authored theorems that have real source ranges and are not projection functions.
module	Module path, e.g., Mathlib.Algebra.AddConstMap.Basic
source_path	Path to the backing .lean file so tools can jump straight into the source tree.
lean4_start/lean4_end	The line where the node's Lean 4 code starts, according to the ConstantInfo. / The line where the node's Lean 4 code ends, decided based on our regex rules.
lean4_code	The raw Lean 4 text sliced from the .lean between lean4_start and lean4_end.
subcategory	Enriched category we assigned based on the category and declaration keyword in lean4_code.
statement	The statement portion of the lean4_code with comments removed.
proof	Complement to the statement, cpaturing the proof chunk using regex.
in_blacklist	Whether the node is on the curated black list.

Table 9. Description of edge fields.

Edge Field	Definition
source_name/target_name	The name entry of source/target node of the edge.
category	Semantic label chosen based on the source and the target's subcategory. Conventions: CONTAINS for file → declaration, IMPORTS for file → file, DEPENDS_ON for anything targeting a theorem/lemma/axiom (including _derived variants), and USES_<CATEGORY> (e.g., USES_CLASS, USES_INSTANCE_DERIVED, USES_STRUCTURE) for everything else.
explicit	Whether a dependency appears in the explicit argument positions of the source declaration. It is True when the referenced constant is used only through explicit parameters(i.e. binderInfo = .default). It is False when the dependency arises solely via implicit arguments, including implicit {} binders or typeclass instance parameters [].
direct	True if the dependency can be obtained from calling function getUsedConstantsAsSet in ConstantInfo). False if it is obtained through further unfolding the direct dependencies of auxiliary lemmas.
could_skip	True if the target node in_blacklist.

Table 10. In-degree distribution by node category (sorted by average out-degree).

Node Categories	#node	avg.	max #in-degree	total #in-degree
class	2,126	348.20	62,237	740,276
structure	3,260	174.95	24,449	570,325
class_derived	11,472	148.51	49,867	1,703,655
instance_derived	1,912	79.71	16,527	152,413
inductive	290	60.77	2,474	17,622
instance	22,462	58.24	32,159	1,308,114
abbrev	2,382	45.66	4,768	108,765
namespace_command	3	39.33	112	118
definition	26,540	32.61	58,468	865,368
section_command	3	30.67	66	92
inductive_component_derived	1,105	14.95	417	16,515
abbrev_derived	226	13.38	382	3,025
structure_derived	9,085	9.52	4,472	86,463
set_option_derived	10	9.00	45	90
definition_derived	15,841	5.77	5,024	91,467
theorem_derived	14,497	5.75	2,388	83,368
theorem	116,096	4.90	3,106	569,172
inductive_derived	1,183	3.65	113	4,315
namespace_command_derived	63	3.27	87	206
lemma	39,832	3.23	1,541	128,737

Table 11. Out-degree distribution by node category (sorted by average out-degree).

Node Categories	#node	avg.	max #out-degree	total #out-degree
file	6,726	43.70	684	293,910
example_derived	1	37.00	37	37
lemma	39,832	31.08	395	1,237,896
theorem	116,096	26.85	474	3,116,776
lemma_derived	3,918	25.36	259	99,369
attribute_derived	3	20.33	25	61
constant_derived	2	20.00	20	40
theorem_derived	14,497	18.66	337	270,442
instance	22,462	17.94	375	403,072
definition_derived	15,841	17.47	117	276,710
definition	26,540	16.62	151	441,037
abbrev_derived	226	14.38	62	3,251
variable_declaration	3	14.00	15	42
abbrev	2,382	13.85	91	32,996
variable_declaration_derived	94	13.67	42	1,285
section_command	3	13.00	27	39
structure_derived	9,085	12.45	135	113,129
instance_derived	1,912	11.94	115	22,828
class_derived	11,472	11.02	123	126,477
set_option_derived	10	10.60	18	106
recursor_derived	48	10.38	23	498
inductive_derived	1,183	8.54	74	10,098
section_command_derived	37	7.32	29	271
namespace_command_derived	63	7.29	20	459
structure	3,260	5.35	84	17,444
inductive	290	4.78	32	1,385
inductive_component_derived	1,105	4.71	40	5,206
namespace_command	3	4.67	9	14
class	2,126	3.87	50	8,231
open_command_derived	34	1.24	17	42
initialize	38	1.05	5	40
constructor	2	0.50	1	1
opaque	107	0.45	6	48
opaque_derived	19	0.42	3	8
syntax	181	0.19	4	35
macro_derived	80	0.11	1	9
syntax_derived	34	0.09	1	3

Table 12. Per-project statistics of the mined Lean Blueprint corpus, sorted by node count. Project names link to the original blueprint.

Project	#Nodes	#Edges	#Formal	#Informal
AlexKontorovich/PrimeNumberTheoremAnd	525	694	487	38
RemyDegenne/brownian-motion	458	732	298	160
teorth/pfr	218	403	214	4
teorth/equational_theories	214	185	145	69
firsching/FormalBook	189	93	55	134
florisvandoorn/carleson	179	213	175	4
Whysoserioushah/BrauerGroup_new	160	18	119	41
YaelDillies/toric	145	213	47	98
thefundamentaltheor3m/Sphere-Packing-Lean	139	201	93	46
kbuzzard/ClassFieldTheory	120	174	55	65
ilpreterosso/LEANearized-RadiiPolynomial	99	158	97	2
emilyriehl/infinity-cosmos	86	104	48	38
leanprover-community/liquid §s1	77	123	77	0
leanprover-community/sphere-eversion	73	101	73	0
Ivan-Sergeyev/seymour	72	137	38	34
b-mehta/ABC-Exceptions	58	89	35	23
mao3j1m0op/algebra-in-lean	54	69	54	0
sinhp/HoTTLean	54	17	17	37
b-mehta/unit-fractions	52	75	47	5
imperialcollegelondon/FLT §ch9	52	43	39	13
cameronfreer/exchangeability	50	69	50	0
florisvandoorn/BonnAnalysis	49	25	14	35
YaelDillies/LeanAPAP	49	61	37	12
ymonbru/M2	49	67	2	47
leanprover-community/flt-regular	45	51	28	17
imperialcollegelondon/FLT §ch10	39	38	39	0
imperialcollegelondon/FLT §ch5	35	40	34	1
james18lpc/GibbsMeasure	35	51	31	4
leanprover-community/liquid §s2	35	53	31	4
vbeffara/Cours_ED	35	0	1	34
AlexandreCharland/BlueprintRep	33	42	28	5
YaelDillies/ChandraFurstLipton	31	49	14	17
imperialcollegelondon/FLT §ch14	30	22	0	30
amatelus/amatelus	27	26	23	4
imperialcollegelondon/FLT §ch13	21	14	17	4
discreteGeometryInLean/polytopes-in-LEAN	18	20	11	7
e-vergo/ChebyshevCircles	18	22	18	0
lean-summer-research/finalized	17	18	17	0
maybemabeline/stone-duality	17	2	5	12
imperialcollegelondon/FLT §ch11	16	21	16	0
rwst/infinite-primes-via-log	16	13	15	1
utensil/lean-ga §s1	16	21	16	0
JonBannon/Junk-Test	13	8	0	13
imperialcollegelondon/FLT §ch8	12	13	12	0
imperialcollegelondon/FLT §ch2	10	11	10	0
imperialcollegelondon/FLT §ch7	10	7	4	6
unaoya/Mills	9	5	4	5
utensil/lean-ga §s2	9	10	8	1
hanwenzhu/LeanArchitect-example	8	8	7	1
imperialcollegelondon/FLT §ch3	8	10	8	0
matijasteblaj/blossoming-principle	8	0	2	6
AdamJasonKern/LBPTester	7	7	7	0
imperialcollegelondon/FLT §ch12	6	5	6	0
Total (53 projects)	**3,805**	**4,651**	**2,728**	**1,077**

premises.

Mathlib4 pretraining itself proceeds in two phases. We first warm up the semantic encoder (Section 4.2), initialized from MGTE (Zhang et al., 2024), with the masked contrastive loss of (Zhu et al., 2025), contrasting independently-encoded queries and premises. We use a cosine learning-rate schedule with warmup ratio 0.04, per-device train batch size 32, gradient accumulation steps 2, and a CachedMNRL mini-batch size of 32, giving an effective per-step batch of $32 \times 16 \times 2 = 1024$.

We then freeze the encoder to train the structural fusion with the multi-positive contrastive loss of Section 4.3 and all three negative-sampling strategies. We use a constant learning-rate schedule with warmup ratio 0.05, per-device train batch size 16, and gradient accumulation steps 8, giving an effective per-step batch of $16 \times 16 \times 8 = 2048$.

Hyperparameters for both phases are listed in Table 13. Detailed training instances are demonstrated in Figures 6 and 7.

Blueprint continual training. The query is encoded from the Blueprint node's informal LaTeX statement only. The candidate premise is encoded from its canonical identifier together with any of the following fields that are present: the informal statement, the Lean declaration name, and the formal Lean source. Reflecting the smaller corpus and lower average out-degree per node, both negative-sampling rates and the dynamic subgraph budget are reduced.

Continual training runs on a single GPU with a cosine learning-rate schedule (warmup ratio 0.05), per-device train batch size 16, and no gradient accumulation, giving an effective per-step batch of 16. Hyperparameters are listed in Table 13. Detailed training instance is demonstrated in Figure 8.

G. Additional Results

G.1. Ablation Studies

Table 14 ablates the components of **DAPS**. We highlight three takeaways.

Structural fusion drives most of the gain. Removing the encoder warm-up costs 5–9 absolute points on Recall@32 across all three benchmarks. Removing structural fusion entirely incurs a much higher cost of 23.33/45.45/18.85 points, respectively. The encoder backbone choice itself becomes critical only under distribution shift: swapping MGTE for a ReProver (Yang et al., 2023) (based on ByT5 (Xue et al., 2022)) preserves most Mathlib performance but collapses **DAPS**-Collection from 24.38 to 1.08, indicating that the base encoder's generalization drives out-of-distribution behavior while structural fusion carries the in-distribution gains.

Multi-hop, bidirectional, all-category subgraphs are needed. Each restriction of the subgraph signal substantially degrades **DAPS**. Reducing the receptive field to 1-hop out-degree only costs 39.81/47.16 absolute points on the two Mathlib benchmarks. Replacing dynamic subgraph sampling (DySS) with uniform random sampling incurs additional performance drops. Restricting the sampled neighbors to the `theorem`, `lemma`, and `definition` categories rather than the full typed schema catastrophically degrades retrieval, confirming that the typed signal carried by class, instance, and structure dependencies is not redundant.

The graph signal itself is essential. Replacing the real Mathlib dependency graph with a dummy graph (random nodes connected by random edges) collapses **DAPS** to 28.93/32.61/6.77 Recall@32 on the two Mathlib benchmarks and **DAPS**-Collection benchmark. This demonstrates the value of **DAPS**'s ability to exploit the typed dependency structure.

G.2. Successful Retrieval Cases

This appendix presents five qualitative retrieval case studies covering all four benchmarks of Section 3.3, with two examples from the most diverse benchmark **DAPS**-Blueprint. Each figure shows the query, the gold positive set, the top-10 retrievals from **DAPS** and the two strongest baselines (LeanHammer and ReProver), and a short commentary. The cases are chosen to illustrate the kinds of structural inferences **DAPS** makes that text-only baselines miss. Failure cases for **DAPS** are documented separately in Appendix G.3.

G.3. Failure Mode Analysis

This appendix documents three distinct failure modes of **DAPS** identified through systematic inspection of low-recall queries across our benchmarks. Table 15 summarizes the taxonomy. The three modes stem from **DAPS**'s structural features, with overgeneralization across different dimensions of the dependency graph: type relations, name modifiers, and topical neighborhoods. We illustrate one representative case per mode below, in the same format as the success cases of Appendix G.2, with the commentary focused on diagnosing the failure mechanism.

Table 13. Training hyperparameters. ℓ = max positives per query (top-ℓ after category-priority sort); m, n = dynamic subgraph sampling budget (random sample/keep after FFN scoring); k_t, k_g, k_c = numbers of temporal/global/cross-batch negatives per query.

	ℓ	m	n	k_t	k_g	k_c
Mathlib4 encoder	32	300	64	0	3	`all`
Mathlib4 structural fusion	32	300	64	30	30	1,500
Blueprint continual training	`all`	33	33	15	45	960

Table 14. Component ablation of **DAPS**. Recall@32 results are reported. Rows progressively remove components from the full configuration (top row) to isolate each component's contribution. Encoder ablations vary the text encoder backbone. Subgraph ablations vary the dependency-graph signal that **DAPS** uses. **DySS**: Dynamic subgraph sampling. **TLD-only**: Restricted node categories of `theorem`, `lemma`, and `definition`. **Mathlib4-R**: Mathlib4-Random. **Mathlib4-H**: Mathlib4-Heldout. **DAPS-Coll.**: **DAPS**-Collection.

Variant	Mathlib4-R	Mathlib4-H	DAPS-Coll.
(a) Full configuration			
DAPS (warmed-up MGTE encoder, DySS, 2-hop bidirectional, all node categories)	**79.63**	**89.31**	**24.38**
(b) Encoder ablations			
− encoder warm-up	74.30	81.08	15.63
− structural fusion (MGTE encoder only)	56.29	43.85	5.53
− MGTE encoder → warmed-up ReProver/ByT5	74.20	81.72	1.08
− MGTE encoder → frozen ReProver/ByT5	59.53	64.87	4.70
(c) Subgraph design ablations			
− 2-hop → 1-hop, out-degree only, with DySS	39.81	42.14	19.84
− DySS → Random sampling, 1-hop bidirectional	46.74	44.49	14.99
− all node categories → TLD-only, random sampling	23.44	25.03	0.72
− real graph → dummy random graph	28.93	32.61	6.77
(d) Contrastive loss ablation			
− group-level contrastive objective → InfoNCE (Oord et al., 2018)	56.48	56.00	15.25

Table 15. Failure modes observed in our case studies, with one representative case per mode (full case studies in Appendices 14–16).

Failure mode	Symptom	Rep. case
Type over-generalization	**DAPS** retrieves typeclass instances on a type that shares the relevant typeclass with the target, but is not the target itself.	App. 14
Lexical sibling drift	When the query is a modifier-prefixed or -suffixed sibling of an in-corpus lemma, **DAPS** retrieves thematically related premises rather than the near-identical sibling.	App. 15
Topical subgraph drift	**DAPS** resolves the correct project, but selects the wrong topical subgraph when several thematically adjacent clusters compete.	App. 16

A training instance for semantic encoder warm up

Query (statement field of Cardinal.mk_mul_le)

```
@[to_additive]
lemma _root_.Cardinal.mk_mul_le : #(s * t) ≤ #s * #t
```

Positive premise (serialized premise text of Set.mul)

```
name: Set.mul

signature: {α : Type u_2} → [Mul α] → Mul (Set α)

lean4_code:
/-- The pointwise multiplication of sets `s * t` and `t` is defined as `{x * y | x ∈ s,
     y ∈ t}` in
locale `Pointwise`. -/
@[to_additive
      /-- The pointwise addition of sets `s + t` is defined as `{x + y | x ∈ s, y ∈ t
    }` in locale
      `Pointwise`. -/]
protected def mul : Mul (Set α) :=
  ⟨image2 (· * ·)⟩

scoped[Pointwise] attribute [instance] Set.mul Set.add
```

Sampled negative premises (3 per query, drawn uniformly from the premise pool, with self and known positives filtered out)

```
name: SimpleGraph.Walk.reverse_transfer

signature: ∀ {V : Type u} {G : SimpleGraph V} {u v : V} (p : G.Walk u v) {H :
    SimpleGraph V} (hp : ∀ (e : Sym2 V), List.instMembership.mem p.edges e → Set.
    instMembership.mem H.edgeSet e), Eq (p.transfer H hp).reverse (p.reverse.transfer H
     ⋯)

lean4_code:
@[simp]
... [truncated]

name: CategoryTheory.Limits.PushoutCocone.epi_inr_of_is_pushout_of_epi

signature: ∀ {C : Type u} [inst : CategoryTheory.Category C] {X Y Z : C} {f : inst.Hom
     X Y} {g : inst.Hom X Z} {t : CategoryTheory.Limits.PushoutCocone f g} (ht :
    CategoryTheory.Limits.IsColimit t) [CategoryTheory.Epi f], Catego
... [truncated]

name: rank_dual_eq_card_dual_of_aleph0_le_rank'

signature: ∀ {K : Type u} [inst : DivisionRing K] {V : Type u_1} [inst_1 :
    AddCommGroup V] [inst_2 : Module K V], Cardinal.instLE.le Cardinal.aleph0 (Module.
    rank K V) → Eq (Module.rank (MulOpposite K) (LinearMap (RingHom.id K) V K)) (
    Cardinal.mk (Line
... [truncated]
```

Figure 6. A training instance for semantic encoder warm up.

A training instance for Mathlib4 pretraining

Query (statement field of Cardinal.mk_mul_le)

```
@[to_additive]
lemma _root_.Cardinal.mk_mul_le : #(s * t) ≤ #s * #t
```

Positive premise (serialized premise text of Set.image2_mul)

```
name: Set.image2_mul

category: theorem

signature: ∀ {α : Type u_2} [inst : Mul α] {s t : Set α}, Eq (Set.image2 (fun (x1 x2 :
     α) => instHMul.hMul x1 x2) s t) (instHMul.hMul s t)

statement: @[to_additive (attr := simp)]
theorem image2_mul : image2 (· * ·) s t = s * t

lean4_code:
@[to_additive (attr := simp)]
theorem image2_mul : image2 (· * ·) s t = s * t :=
  rfl
```

Sampled negative premises (3 shown from sample_negatives (random_hard=30, global=30; positives/self filtered))

```
name: Set.natCard_neg

category: lemma_derived

signature: ∀ {G : Type u_1} [inst : InvolutiveNeg G] (s : Set G), Eq (Nat.card (Set.
    neg.neg s).Elem) (Nat.card s.Elem)

statement: @[to_additive]
lemma natCard_inv (s : Set G) : Nat.card ↑(s⁻¹) = Nat.card s

lean4_code:
@[to_additive]
lemma natCard_inv (s : Set
... [truncated]

name: Set.ncard_smul_set

category: lemma

signature: ∀ {G : Type u_1} {α : Type u_3} [inst : Group G] [inst_1 : MulAction G α] (
    a : G) (s : Set α), Eq (instHSMul.hSMul a s).ncard s.ncard

statement: @[to_additive (attr := simp)]
lemma ncard_smul_set (a : G) (s : Set α) : (a • s).ncard = s.ncard

lean4_code:

... [truncated]

name: Set.natCard_smul_set

category: lemma

signature: ∀ {G : Type u_1} {α : Type u_3} [inst : Group G] [inst_1 : MulAction G α] (
    a : G) (s : Set α), Eq (Nat.card (instHSMul.hSMul a s).Elem) (Nat.card s.Elem)

statement: @[to_additive]
lemma natCard_smul_set (a : G) (s : Set α) : Nat.card ↑(a • s) = Na
... [truncated]
```

Figure 7. A training instance for Mathlib4 pretraining.

A training instance for Blueprint continual training

Query (statement field of equational_theories/partial-1323-sound)

```
Let \({\mathcal F}\) be a partial solution. Then if one defines a partial operation \(\
    diamond \) on \(M\) by requiring axioms (i), (ii), imposing the additional
    operations \[ (\phi _a(b)^n x,a) \diamond (\phi _b(c)^n y,b) = (\phi _c(a)^n z,c)
    \]
```

Positive premise (serialized premise text of equational_theories/partial-1323)

```
id: partial-1323

nlp: A partial solution is a finite family \({\mathcal F}\) of tuples \((x,y,z,a,b,c) \
    in (\mathbb {Q}^\times )^3 \times G^6\) with \(a,b,c,0\) distinct, such that the
    tuples \((\phi _a(b)^n x,a, \phi _b(c)^n y,b)\), \((\phi _b(c)^n y,b,\phi _c(a)^n z
    ,c)\), \((\phi _c(a)^n z,c,\phi _a(b)^{n+1} x, a)\) for \((x,y,z,a,b,c) \in {\
    mathcal F}\) and \(n \geq 0\) are all distinct.

lean_decl_name: partial-1323
```

Sampled negative premises (3 shown from sample_negatives (repo=15, global=45; positives/self filtered))

```
id: eq40

nlp: Equation 40 is the law \(0 \diamond 0 \simeq 1 \diamond 1\) (or the equation \(x \
    diamond x = y \diamond y\)).

lean_decl_name: Equation40

lean4:
def Equation40 (G: Type*) [Magma G] := ∀ x y : G, x ◇ x =
... [truncated]

id: asterix-obelix-finite

nlp: Any finite magma satisfying the Asterix law (E65) also is left-cancellative and
    satisfies the Obelix law (E1491).

lean_decl_name: asterix-obelix-finite

id: greedy-iterate

nlp: Every partial solution can be extended to a complete solution that satisfies
    Equation 2 and Equation 3, and hence 1323.

lean_decl_name: Eq1323.exists_complete_function

lean4:
theorem exists_com
... [truncated]
```

Figure 8. A training instance for Blueprint continual training.

Case study: `pfr/multidist-ruzsa-III` (DAPS-Blueprint)

Query (informal statement):
Let $m \geq 2$ and let $X_{[m]}$ be a tuple of G-valued random variables. If the X_i all have the same distribution, then $D[X_{[m]}] \leq m \cdot d[X_i; X_i]$ for any $1 \leq i \leq m$.

Gold positives (6): `pfr/ruz-copy`, `pfr/ruz-indep`, `pfr/sumset-lower`, `pfr/klm-1`, `pfr/multidist-copy`, `pfr/multidist-indep`.

DAPS top-10:
1. `pfr/ruzsa-diff`
2. ✓ `pfr/ruz-copy`
3. ✓ `pfr/ruz-indep`
4. `pfr/ruzsa-triangle`
5. `pfr/multidist-nonneg`
6. `pfr/sym-zero`
7. ✓ `pfr/sumset-lower`
8. `pfr/multidist-def`
9. `pfr/ruzsa-nonneg`
10. `pfr/multidist-ruzsa-I`

Recall@32: ***5/6*** *(0.83)*

LeanHammer top-10:
1. `PrimeNumberTheoremAnd/fks2-theorem-3`
2. `FormalBook/ch20theorem2`
3. `FormalBook/ch16theorem2`
4. `pfr/foursum-bound`
5. `LeanAPAP/drc`
6. `PrimeNumberTheoremAnd/bklnw-cor-3-1`
7. `pfr/phi-first-estimate`
8. `PrimeNumberTheoremAnd/buthe-theorem-2f`
9. `PrimeNumberTheoremAnd/buthe-theorem-2d`
10. `pfr/ruzsa-diff`

Recall@32: ***0/6*** *(0.00)*

ReProver top-10:
1. `FormalBook/ch38lemma2`
2. `carleson/antichain-decomposition`
3. `pfr/gen-ineq`
4. `ABC-Exceptions/prop:ThueEquations`
5. `toric/1-1-phiA`
6. `polytopes-in-LEAN/prop:v_is_h`
7. `PrimeNumberTheoremAnd/Dedekind-nonvanishing`
8. `FormalBook/ch22lemma1`
9. `PrimeNumberTheoremAnd/Dedekind-PNT`
10. `equational_theories/axiom-b`

Recall@32: ***0/6*** *(0.00)*

Commentary. **DAPS** retrieves `ruz-copy` and `ruz-indep`, the two defining lemmas linking Ruzsa distance $d[\cdot;\cdot]$ to entropies, in positions 2 and 3, and surfaces 5 of the 6 gold premises within its top-16. LeanHammer and ReProver miss *all* 6 gold premises within their top-32, instead of retrieving lemmas from unrelated Blueprint projects (`FormalBook`, `PrimeNumberTheoremAnd`, `carleson`, `ABC-Exceptions`, `toric`). The gold premises form a tight subgraph in PFR's Ruzsa-distance section. **DAPS**'s typed dependency edges keep retrieval inside this neighborhood, while the text-only baselines are pulled across the corpus by surface similarity to other entropy and inequality lemmas.

Figure 9. Case study from the `teorth/pfr` Blueprint project. ✓ marks premises in the gold positive set. Premise names follow the PFR Blueprint label scheme.

Case study: `PrimeNumberTheoremAnd/fks-corollary-3-12` (DAPS-Blueprint)

Query (informal statement):

Let $5/8 < \sigma_2 \leq 1$, $t_0 = t_0(\sigma_2, x) = \max\left(H_{\sigma_2}, \exp\left(\sqrt{\frac{\log x}{R}}\right)\right)$, $T > t_0$. *Let* $K \geq 2$, $\lambda = (T/t_0)^{1/K}$, *and consider* $(t_k)_{k=0}^{K}$ *the sequence given by* $t_k = t_0\lambda^k$. *Then*

$$\Sigma^1_{\sigma_2} = 2 \sum_{\substack{0<\gamma<T \\ \sigma_2 \leq \beta < 1}} \frac{x^{\beta-1}}{\gamma} \leq \varepsilon_4(x, \sigma_2, K, T),$$

Gold positives (4): `PrimeNumberTheoremAnd/zero-density-bound`, `PrimeNumberTheoremAnd/Riemann-von-Mangoldt-estimate`, `PrimeNumberTheoremAnd/RH-up-to`, `PrimeNumberTheoremAnd/classical-zero-free-region`.

DAPS top-10:

1. ✓ `PrimeNumberTheoremAnd/zero-density-bound`
2. ✓ `PrimeNumberTheoremAnd/Riemann-von-Mangoldt-estimate`
3. ✓ `PrimeNumberTheoremAnd/RH-up-to`
4. `PrimeNumberTheoremAnd/Etheta-def`
5. `PrimeNumberTheoremAnd/bklnw-eq_A_16`
6. `ABC-Exceptions/lem:SubSumGeometry`
7. `PrimeNumberTheoremAnd/Epsi-def`
8. `ABC-Exceptions/lem:Trivialaibici`
9. `PrimeNumberTheoremAnd/classical-bound-psi`
10. `Sphere-Packing-Lean/thm:ramanujan-formula`

Recall@32: ***4/4*** *(1.00)*

LeanHammer top-10:

1. `PrimeNumberTheoremAnd/fks-proposition-3-11`
2. `PrimeNumberTheoremAnd/fks-proposition-3-14`
3. `PrimeNumberTheoremAnd/fks2-lemma-10b`
4. `PrimeNumberTheoremAnd/bklnw-eq_A_13`
5. `PrimeNumberTheoremAnd/bklnw-eq_A_10`
6. `PrimeNumberTheoremAnd/fks2-corollary-26`
7. `PrimeNumberTheoremAnd/bklnw-cor-3-1`
8. `carleson/Holder-van-der-Corput`
9. `PrimeNumberTheoremAnd/bklnw-eq_A_12`
10. `PrimeNumberTheoremAnd/buthe-theorem-2d`

Recall@32: ***0/4*** *(0.00)*

ReProver top-10:

1. `toric/1-2-14-dual-rat-cone`
2. `ABC-Exceptions/def:nu`
3. `PrimeNumberTheoremAnd/lem:abadimpseri`
4. `PrimeNumberTheoremAnd/bklnw-thm_16`
5. `PrimeNumberTheoremAnd/lem:aachcanc`
6. `FormalBook/ch26lemma_d`
7. `PrimeNumberTheoremAnd/bklnw-prop-3-sub-1`
8. `PrimeNumberTheoremAnd/bklnw-cor-5-1`
9. `PrimeNumberTheoremAnd/lem:abadeulmac'`
10. `PrimeNumberTheoremAnd/symm_inv_log`

Recall@32: ***0/4*** *(0.00)*

Commentary. DAPS retrieves *all four* gold premises in its top-32 (positions 1, 2, 3, and 16), giving perfect recall. The two failure modes of the baselines are revealing. LeanHammer stays inside `PrimeNumberTheoremAnd` but is captured by surface similarity to the query name: it returns `fks-proposition-3-11/14` and several `bklnw-eq_*` entries from neighboring sections of the same paper, missing the gold premises that live in the earlier zero-density and zero-free-region sections. ReProver drifts further, retrieving lemmas from `toric` and `ABC-Exceptions` entirely outside the relevant subgraph. **DAPS**'s typed dependency edges resolve the precise mathematical neighborhood of zero-density estimates and zero-free regions of ζ, that this corollary actually consumes.

Figure 10. Case study from the `AlexKontorovich/PrimeNumberTheoremAnd` Blueprint project. ✓ marks premises in the gold positive set. Premise names follow the PFR Blueprint label scheme.

Case study: `lean_workbook_plus_39047` (cosine difference formula, DAPS-Collection)

Query (Lean 4 statement):

```
theorem lean_workbook_plus_39047 (x y : R) :
    cos (x - y) = cos x * cos y + sin x * sin y := by
```

Gold positives (16, ranked by priority): `Real.cos`, `Real.instSub`, `Real.instAdd`, `Real.cos_sub`, `Real.sin`, `AddGroup`, `neg_add_cancel`, `add_neg_cancel`, `lt_trans`, `IsStrictOrderedRing`, ... (16 total).

DAPS top-10:
1. `Real.instRatCast`
2. ✓ `Real.instSub`
3. `Real.pseudoMetricSpace`
4. `Real.instMul`
5. ✓ `Real.instAdd`
6. `Real.instRing`
7. `Real.instNeg`
8. `Real.instDivisionRing`
9. `Real.instSemigroup`
10. `Real.normedCommRing`

Recall@32: ***4/16*** *(0.25)*

LeanHammer top-10:
1. `Complex.cos`
2. ✓ `Real.cos`
3. `Complex.sin`
4. `Complex.cos_neg`
5. ✓ `Real.sin`
6. `Complex.cos_add`
7. `Complex.ofReal_cos`
8. `Complex.ofReal_sin`
9. `Complex.sin_add`
10. `Real.cos_add`

Recall@32: ***3/16*** *(0.19)*

ReProver top-10:
1. ✓ `Real.cos`
2. `Complex.ofReal_cos`
3. `Complex.sin`
4. `Complex.cos`
5. `Complex.ofReal_sin`
6. `Complex.sin_eq_zero_iff_cos_eq`
7. `Complex.cos_eq_zero_iff_sin_eq`
8. `Complex.sin_ofReal_re`
9. `Real.cos_zero`
10. ✓ `Real.sin`

Recall@32: ***2/16*** *(0.12)*

Commentary. All three selectors hit at least one gold premise here, but the top-10 reveals different retrieval behaviors. **DAPS**'s top-10 is dominated by `Real.inst*` typeclass instances (8 of 10 entries), which is the algebraic infrastructure on which the cosine-difference proof actually rewrites. LeanHammer surfaces both `Real.cos` and `Real.sin` in its top-5 but wastes the remaining slots on `Complex.*` duplicates (6 of 10 entries) of the same trigonometric lemmas. ReProver shows the same `Complex.*` drift more severely. The two signals are complementary: textual encoders catch the namesake trig lemmas faster, but **DAPS**'s typed dependency edges surface the typeclass and structural premises that proofs of competition-style theorems also depend on, yielding the highest overall Recall@32.

Figure 11. Case study from the **DAPS**-Collection benchmark (a Lean Workbook problem). ✓ marks premises in the gold positive set. Premise names follow Mathlib4 declaration paths.

Case study: `add_lt_add_of_lt_of_lt` (Mathlib4-Heldout)

Query (Lean 4 source, additive version generated by `@[to_additive]`):

```
@[to_additive (attr := gcongr high)]
theorem mul_lt_mul_of_lt_of_lt [MulLeftStrictMono α]
    [MulRightStrictMono α]
    {a b c d : α} (h₁ : a < b) (h₂ : c < d) :
    a * c < b * d
```

Gold positives (6): `add_lt_add_left`, `add_lt_add_right`, `instTransLt_mathlib`, `AddLeftStrictMono`, `AddRightStrictMono`, `Preorder`.

DAPS top-10:
1. ✓ `Preorder`
2. `MulLeftStrictMono`
3. `MulRightStrictMono`
4. ✓ `AddRightStrictMono`
5. ✓ `AddLeftStrictMono`
6. `mul_lt_mul_of_lt_of_lt`
7. ✓ `add_lt_add_left`
8. ✓ `add_lt_add_right`
9. `mul_lt_mul_left'`
10. `mul_lt_mul_right'`

Recall@32: ***6/6*** *(1.00)*

LeanHammer top-10:
1. `gauge`
2. `gauge_def`
3. `gauge_def'`
4. `Set.instMembership`
5. `Module`
6. `gauge_nonneg`
7. `Real.instInfSet`
8. `gauge_zero`
9. `Set`
10. `InfSet.sInf`

Recall@32: ***0/6*** *(0.00)*

ReProver top-10:
1. `min_lt_max_of_mul_lt_mul`
2. `Left.min_le_max_of_mul_le_mul`
3. `Right.min_le_max_of_mul_le_mul`
4. `LinearOrder.lift`
5. `max_mul_min`
6. `max_min_distrib_right`
7. `max_min_distrib_left`
8. `min_le_max_of_mul_le_mul`
9. `fn_max_mul_fn_min`
10. `min_mul_max`

Recall@32: ***0/6*** *(0.00)*

Commentary. DAPS retrieves all six gold premises within its top-11 and five of them within its top-10. The remaining four entries in **DAPS**'s top-10 are the multiplicative siblings (`MulLeftStrictMono`, `MulRightStrictMono`, `mul_lt_mul_of_lt_of_lt`, `mul_lt_mul_left'`, `mul_lt_mul_right'`), which are the `@[to_additive]` source declarations from which the additive query is generated. LeanHammer drifts to the unrelated `gauge` namespace (a functional analysis context) and loses the strict monotonicity meaning entirely. ReProver stays closer in spirit by retrieving `min`/`max` lemmas about multiplication, but still misses all six gold premises. The dependency graph captures the `@[to_additive]` inheritance link and lifts both the multiplicative parent and the additive children into top positions, while text-only retrieval reaches for surface similarity and ends up in adjacent but unrelated parts of Mathlib.

Figure 12. Case study from the Mathlib4-Held-out benchmark. ✓ marks premises in the gold positive set. Premise names are Mathlib4 declaration paths.

Case study: `Algebra.map_leftMulMatrix_localization` (Mathlib4-Random)

Query (Lean 4 statement):

```
theorem Algebra.map_leftMulMatrix_localization
    {ι : Type*} [Fintype ι] [DecidableEq ι]
    (b : Basis ι R S) (a : S) :
    (algebraMap R Rₘ).mapMatrix (leftMulMatrix b a) =
    leftMulMatrix (b.localizationLocalization Rₘ M Sₘ)
                  (algebraMap S Sₘ a)
```

Gold positives (32, ranked by priority): `Algebra.leftMulMatrix_eq_repr_mul`, `Matrix.ext`, `Module.Basis.localizationLocalization_apply`, `Module.Basis.localizationLocalization_repr_algebraMap`, `Algebra.algebraMapSubmonoid`, `Algebra.leftMulMatrix`, `Matrix`, `Module.Basis.localizationLocalization`, `RingHom.mapMatrix`, `algebraMap`, ... (32 total).

DAPS top-10:
1. ✓ `Algebra.algebraMapSubmonoid`
2. ✓ `Module.Basis.localizationLocalization`
3. `IsScalarTower.algebraMap_apply`
4. ✓ `Module.Basis`
5. `Module.Basis.coe_mapCoeffs`
6. `RingEquiv.ofBijective`
7. ✓ `Module.Basis.instFunLike`
8. ✓ `Algebra.leftMulMatrix`
9. `Algebra.smul_def`
10. `Algebra.toModule`

Recall@32: ***11/32*** *(0.34)*

LeanHammer top-10:
1. `WeierstrassCurve.Affine.Point.map`
2. `WeierstrassCurve.Affine.Point`
3. `WeierstrassCurve.Affine.Point.map_injective`
4. `WeierstrassCurve.Affine.Point.baseChange`
5. `WeierstrassCurve.Affine.Point.instAddZeroClass`
6. `WeierstrassCurve.Affine.Point.instZero`
7. `WeierstrassCurve.Affine`
8. `WeierstrassCurve.Affine.Point.instAddCommGroup`
9. `Algebra`
10. `WeierstrassCurve.toAffine`

Recall@32: ***3/32*** *(0.09)*

ReProver top-10:
1. `ModuleCat.isFG`
2. `ModuleCat.forget_obj`
3. `LinearMap.id_fgModuleCat_comp`
4. `FGModuleCat.hom_id`
5. `CategoryTheory.Linear.fullSubcategory`
6. `LinearMap.comp_id_fgModuleCat`
7. `FGModuleCat.instFullModuleCatForget`
8. `ModuleCat.hom_id`
9. `ModuleCat.Algebra.instLinear`
10. `FGModuleCat.carrier`

Recall@32: ***0/32*** *(0.00)*

Commentary. DAPS retrieves five gold premises in its top-10 (`Algebra.algebraMapSubmonoid`, `Module.Basis.localizationLocalization`, `Module.Basis`, `Module.Basis.instFunLike`, `Algebra.leftMulMatrix`) and 11 of the 32 gold premises within its top-32. The remaining entries cluster tightly in the `Algebra` and `Module.Basis` namespaces that the query actually depends on. LeanHammer drifts to `WeierstrassCurve.Affine.Point` (elliptic curve points), losing the algebra and localization context entirely. ReProver drifts to `ModuleCat` and `FGModuleCat` (the category-theoretic recasting of module theory), retrieving zero gold premises in its top-32. **DAPS**'s typed dependency edges resolve the precise mathematical neighbourhood that this theorem consumes, while text-only baselines latch onto unrelated parts of Mathlib that share only superficial vocabulary.

Figure 13. Case study from the Mathlib4-Random benchmark. ✓ marks premises in the gold positive set. Premise names are Mathlib4 declaration paths.

Failure case study: `paul_2_mod_5` (DAPS-Collection)

Query (Lean 4 statement):

```
theorem paul_2_mod_5 (p : ℕ) (hp : p.Prime) (hp1 : p ≠ 5) :
    p % 5 ≠ 0 := by
```

Gold positives (17, ranked by priority): `Nat.instMulZeroClass`, `Nat.Prime`, `Nat.instOne`, `Nat.instCharZero`, `Nat.Prime.eq_one_or_self_of_dvd`, `Nat.instAddMonoidWithOne`, `AddGroup`, `neg_add_cancel`, `add_neg_cancel`, `lt_trans`, ... (17 total).

DAPS top-10:
1. `Rat.instNormedField`
2. `Rat.nontrivial`
3. `mul_ne_zero`
4. `Rat.instCharZero`
5. `Rat.instField`
6. `Fact.elim`
7. `NormMulClass.toNoZeroDivisors`
8. `Rat.mul_den`
9. `Real.instNontrivial`
10. `Nat.Prime.ne_zero`

*Recall@32: **0/17** (0.00)*

LeanHammer top-10:
1. ✓ `Nat.Prime`
2. `Nat.decidablePrime`
3. `Nat.Prime.ne_one`
4. `Nat.Prime.two_le`
5. `Nat.Prime.pos`
6. `Nat.Prime.one_le`
7. `Nat.Prime.one_lt`
8. `Nat.Prime.ne_zero`
9. `Nat.Prime.not_dvd_one`
10. `Nat.Prime.prime`

*Recall@32: **2/17** (0.12)*

ReProver top-10:
1. `Nat.Prime.ne_one`
2. ✓ `Nat.Prime`
3. `Nat.Prime.pos`
4. `Nat.Prime.one_lt`
5. `Nat.prime_def`
6. `Nat.prime_def_lt`
7. `Nat.Prime.ne_zero`
8. ✓ `Nat.Prime.eq_one_or_self_of_dvd`
9. `Nat.prime_def_lt'`
10. `Nat.Prime.pred_pos`

*Recall@32: **2/17** (0.12)*

Commentary. This is a failure mode for **DAPS**. The query is a short fact about natural-number primality, but **DAPS**'s top-10 is dominated by typeclass instances on `Rat` and `Real` (`Rat.instNormedField`, `Rat.instField`, `Rat.instCharZero`, `Real.instNontrivial`) rather than the corresponding `Nat` instances that the proof actually rewrites against. **DAPS** likely over-generalizes along the typeclass dimension. Several gold premises are typeclass instances that both `Nat` and `Rat` satisfy (`instCharZero`, `Field`, `IsStrictOrderedRing`), and **DAPS**'s structural features apparently collapse the two types into the same neighborhood, then pick the more densely-connected `Rat` variants. Text-only baselines avoid this confusion here because the query name `Nat.Prime` is lexically explicit. LeanHammer and ReProver both retrieve `Nat.Prime` and several `Nat.Prime.*` lemmas, and ReProver also surfaces the gold theorem `Nat.Prime.eq_one_or_self_of_dvd` at position 8. All three selectors still miss most of the typeclass infrastructure that the proof depends on, so absolute Recall@32 remains low for everyone. The case illustrates a real limitation of dependency-aware retrieval. When several types share the relevant typeclass structure, and the query carries a strong text signal toward only one of them, structural features alone can pick the wrong type.

Figure 14. Failure case study from **DAPS**-Collection (a Lean Workbook problem). ✓ marks premises in the gold positive set. **DAPS** retrieves zero gold premises in its top-32 on this query, while LeanHammer and ReProver each retrieve two. Premise names are Mathlib4 declaration paths.

Failure case study: `brownian-motion/lem:M_uniform_integrable` (DAPS-Blueprint)

Query (informal statement):
The sequence $(M_T^n)_{n\in\mathbb{N}}$ is uniformly integrable (bounded in L^1 norm).

Gold positives (2): `brownian-motion/lem:A_uniform_integrable`, `brownian-motion/def:classD`.

DAPS top-10:
1. `BonnAnalysis/lem:fourier-multiplication`
2. `exchangeability/lem:covariance_structure`
3. `brownian-motion/lem:uniformIntegrableIffNorm`
4. `PrimeNumberTheoremAnd/RectangleIntegral_tendsTo_VerticalIntegral`
5. `PrimeNumberTheoremAnd/ResidueTheoremOnRectangleWithSimplePole`
6. `BonnAnalysis/thm:kernel-approximation`
7. `PrimeNumberTheoremAnd/HolomorphicOn.vanishesOnRectangle`
8. `brownian-motion/def:elemStochIntegralBilin`
9. `exchangeability/lem:contractable_dist_eq`
10. `brownian-motion/lem:uniformIntegrableDominatedSingleton`

Recall@32: ***0/2*** *(0.00)*

LeanHammer top-10:
1. `brownian-motion/lem:uniformIntegrableNorm`
2. ✓ `brownian-motion/lem:A_uniform_integrable`
3. `brownian-motion/lem:Submartingale.uniformIntegrable_of_antitone_of_ge`
4. `brownian-motion/lem:uniformIntegrableIffNorm`
5. `brownian-motion/lem:uniformIntegrableDominatedSingleton`
6. `brownian-motion/lem:uniformIntegrableAdd`
7. `brownian-motion/def:HasIntegrableSup`
8. `brownian-motion/lem:uniformIntegrableDominated`
9. `radii-polynomial/thm:norm_mul_le`
10. `brownian-motion/lem:covariation_eq_inner`

Recall@32: ***1/2*** *(0.50)*

ReProver top-10:
1. ✓ `brownian-motion/lem:A_uniform_integrable`
2. `brownian-motion/lem:uniformIntegrableNorm`
3. `brownian-motion/lem:IsSquareIntegrable.sup_eLpNorm_eq_eLpNorm_limitProcess`
4. `ClassFieldTheory/lem:norm_comm`
5. `FLT_ch5/Hurwitz.norm`
6. `brownian-motion/lem:Submartingale.uniformIntegrable_of_antitone_of_ge`
7. `FLT_ch5/Hurwitz.norm_one`
8. `FLT_ch5/Hurwitz.exists_near`
9. `FLT_ch5/Hurwitz.norm_zero`
10. `brownian-motion/cor:doob_lp_norm`

Recall@32: ***1/2*** *(0.50)*

Commentary. This is a second failure mode for **DAPS**, distinct from the failure in Appendix 14. The query has only two gold premises. The first one, `A\_uniform\_integrable`, differs from the query name `M\_uniform\_integrable` by a single letter in the prefix, making it a strong lexical match that text-only baselines exploit directly. LeanHammer surfaces it at position 2 and ReProver at position 1. **DAPS** instead drifts across the Blueprint corpus, retrieving thematically related premises about analysis and integrability from unrelated projects (`BonnAnalysis/lem:fourier-multiplication`, `PrimeNumberTheoremAnd/RectangleIntegral_tendsTo_VerticalIntegral`, `exchangeability/lem:covariance_structure`). **DAPS**'s structural features over-generalize across the global corpus when the query carries a strong lexical anchor that points to a tightly-named in-project sibling. Together with the type-level over-generalization illustrated in Appendix 14, this case suggests that **DAPS** would benefit from a stronger text-side anchoring signal to suppress structural neighbors that share thematic context but live in distant parts of the corpus.

Figure 15. Failure case study from **DAPS**-Blueprint (the brownian-motion project). ✓ marks premises in the gold positive set. **DAPS** retrieves zero gold premises in its top-32 here, while LeanHammer and ReProver each retrieve one (the lexically near-identical `A_uniform_integrable`). The case illustrates a second failure mode for **DAPS**. When the gold premise shares strong lexical similarity with the query name, text-only baselines exploit the direct match, whereas **DAPS** drifts across the global Blueprint corpus along thematic lines.

Failure case study: `PrimeNumberTheoremAnd/ResidueOfTendsTo` (DAPS-Blueprint)

Query (informal statement):

If a function f is holomorphic in a neighborhood of p and $\lim_{s\to p}(s-p)\,f(s) = A$, then $f(s) = \dfrac{A}{s-p} + O(1)$ near p.

Gold positives (1): `PrimeNumberTheoremAnd/existsDifferentiableOn_of_bddAbove`.

DAPS top-10:

1. `PrimeNumberTheoremAnd/tendsto_rpow_atTop_nhds_zero_of_norm_gt_one`
2. `PrimeNumberTheoremAnd/MellinConvolutionTransform`
3. `radii-polynomial/prop:fixed_point_iff_zero`
4. `PrimeNumberTheoremAnd/DeltaSpikeMass`
5. `radii-polynomial/thm:neumann_series`
6. `PrimeNumberTheoremAnd/MellinOfPsi`
7. `radii-polynomial/prop:operator_norm_properties`
8. `PrimeNumberTheoremAnd/MellinOfDeltaSpike`
9. `radii-polynomial/lem:newton_fderiv`
10. `carleson/nontangential-operator-boundary`

Recall@32: ***0/1*** *(0.00)*

LeanHammer top-10:

1. `PrimeNumberTheoremAnd/isHolomorphicOn`
2. `PrimeNumberTheoremAnd/HolomorphicOn_riemannZeta0`
3. `PrimeNumberTheoremAnd/logDerivResidue`
4. `PrimeNumberTheoremAnd/LogDerivZetaHolcSmallT`
5. `PrimeNumberTheoremAnd/ResidueMult`
6. ✓ `PrimeNumberTheoremAnd/existsDifferentiableOn_of_bddAbove`
7. `PrimeNumberTheoremAnd/LogDerivZetaHolcLargeT`
8. `PrimeNumberTheoremAnd/schwarz-id`
9. `sphere-eversion/lem:param_trick`
10. `PrimeNumberTheoremAnd/BddAbove_to_IsBigO`

Recall@32: ***1/1*** *(1.00)*

ReProver top-10:

1. `FormalBook/ch26lemma_d`
2. `M2/def:homotopy_diagram`
3. `M2/def:homotopy_colimit`
4. `FormalBook/ch41proof1`
5. `FormalBook/ch41proof5`
6. `FormalBook/ch41proof4`
7. `FormalBook/ch41proof3`
8. `FormalBook/ch41proof2`
9. `Sphere-Packing-Lean/lemma:phi0-transform`
10. `ABC-Exceptions/def:nu`

Recall@32: ***0/1*** *(0.00)*

Commentary. This case illustrates a within-project topical drift for **DAPS**. The query is a residue-style statement about holomorphic functions, and the unique gold premise is `existsDifferentiableOn_of_bddAbove`, a technical lemma in PNT's holomorphy/residue cluster. **DAPS** stays inside the `PrimeNumberTheoremAnd` and `radii-polynomial` projects but drifts to a different topical subgraph, retrieving the Mellin-transform and operator-norm lemmas (`MellinConvolutionTransform`, `MellinOfPsi`, `MellinOfDeltaSpike`, `neumann_series`, `operator_norm_properties`). LeanHammer's text encoder identifies the correct PNT subcluster (`isHolomorphicOn`, `HolomorphicOn_riemannZeta0`, `logDerivResidue`, `ResidueMult`, `BddAbove_to_IsBigO`) and surfaces the gold at position 6. ReProver scatters across unrelated projects (`FormalBook`, `M2`, `Sphere-Packing-Lean`, `ABC-Exceptions`) and also fails. The case shows that even when **DAPS** correctly localizes to the right project, its structural features can pick the wrong topical subgraph within that project when several thematically adjacent clusters compete.

Figure 16. Failure case study from **DAPS**-Blueprint (the PrimeNumberTheoremAnd project). ✓ marks premises in the gold positive set. **DAPS** retrieves zero gold premises in its top-32, while LeanHammer retrieves the single gold premise at position 6, and ReProver also fails. The case reveals a within-project topical drift for **DAPS**. Structural features can resolve the right project but pick the wrong topical subgraph when several thematically adjacent clusters exist.