GARNET: GRAPH-AUGMENTED RERANKING FOR PREMISE RETRIEVAL IN LEAN

Anonymous authors
Paper under double-blind review

ABSTRACT

Premise retrieval is a central component of LLM-based theorem proving: a prover must identify useful lemmas from a library containing hundreds of thousands of declarations. Most existing retrievers rely on direct matching with either a natural-language query or a local proof state, ignoring the structural relationships encoded in the library's dependency graph. We propose GARNET, a graph-augmented reranking framework for Lean that expands query- and state-conditioned anchor sets along local dependency neighborhoods and reranks candidates by combining direct retrieval evidence with structural context. On end-to-end retrieval benchmarks, GARNET improves R@20 by over 20 absolute points against strong baselines on Mathlib 4.28 and maintains strong performance on Lean Workbook. When integrated into an LLM-based proving agent for proof-sketch completion on a curated 60-theorem subset of ProofNet-Verified, it achieves $77.7 \pm 3.0\%$ mean single-attempt accuracy and 88.3% pass@5, the highest among all evaluated systems, with a substantial margin over learned state-only retrievers. More broadly, our results show that effective premise retrieval benefits from combining complementary query- and state-conditioned evidence with structural relationships among library declarations.

1 INTRODUCTION

Large language models are increasingly integrated with interactive theorem provers such as Lean (Moura & Ullrich, 2021), where proving non-trivial theorems often requires identifying useful declarations from large formal libraries such as Mathlib (The mathlib Community, 2020). Recent retrieval-augmented proving systems show that retrieving useful premises can reduce the reasoning burden of the language model and improve proof generation (Gao et al., 2026; Varambally et al., 2026; Xu et al., 2026). Analyses of Lean-based agents further indicate that proof success depends on whether the underlying library contains suitable prerequisites (Zhang et al., 2026). Improving premise retrieval is therefore essential to the scalability and reliability of automated theorem proving.

Current premise retrievers rely mainly on direct matching with either the agent query or the current proof state. Query-based retrievers rank declarations by their semantic similarity to the agent's proof intent (Gao et al., 2026), while state-based retrievers identify declarations that are relevant to the local formal context (Yang et al., 2023; Petrovčič et al., 2025; Zhu et al., 2026; Tao et al., 2025). These signals are complementary: the query conveys the intended mathematical idea, whereas the proof state provides the formal context needed to use a premise. This motivates combining query- and state-based retrieval to improve premise selection.

Moreover, query-based retrieval may return a theorem that matches the agent's intended mathematical idea but is not directly usable in the proof. The proof may instead rely on one of the theorem's dependencies, which direct matching can miss even though it is structurally connected to the retrieved theorem in Mathlib's dependency graph. This suggests that the dependency structure can expose useful premises that direct query matching overlooks.

Based on this insight, we propose GARNET, a graph-augmented reranking framework for Lean theorem proving. As illustrated in Figure 1, GARNET combines complementary retrieval signals from semantic queries and proof states with local exploration of the directed Mathlib dependency graph.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

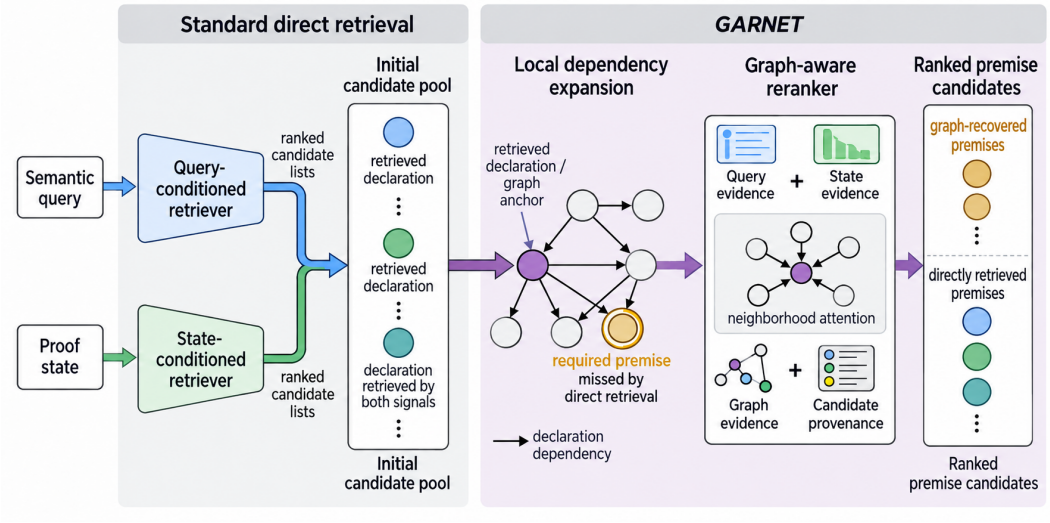


Figure 1: Overview of GARNET. Standard direct retrieval independently retrieves candidate declarations from the semantic query and the current proof state. The retrieved declarations are then unified and used both as premise candidates and as anchors for local dependency-graph expansion. The resulting candidate pool is scored by a graph-aware reranker that combines direct retrieval evidence with dependency-neighborhood information and candidate provenance.

The retrieved declarations serve as high-precision anchors for dependency-guided candidate expansion, while a learned reranker combines direct retrieval evidence with graph-derived neighborhood information to prioritize the resulting candidates.

Our contributions are threefold. **(1) A graph-augmented framework for Lean premise retrieval.** GARNET combines query- and state-conditioned retrieval with dependency-guided candidate expansion and graph-aware reranking. **(2) Strong retrieval performance across evaluation settings.** Across Mathlib 4.28 and Lean Workbook, GARNET improves R@20 over the strongest query- or state-only baseline by 23.7–32.4 absolute points. Ablations show that both dependency expansion and graph-aware reranking contribute to these gains. **(3) End-to-end evaluation in an LLM proving agent.** We introduce SketchFill-60, a controlled proof-sketch completion setting derived from ProofNet-Verified. With adaptive use of query- and state-conditioned retrieval, GARNET achieves $77.7 \pm 3.0\%$ mean single-attempt accuracy and 88.3% pass@5, the highest among all evaluated systems, with a substantial margin over learned state-only retrieval systems.

2 RELATED WORK

2.1 PREMISE SELECTION

Existing premise retrieval systems for Lean primarily differ in the signal used to condition retrieval. Query-based methods retrieve declarations from a theorem statement or a natural-language description: Loogle performs syntactic search over the library (Lean FRO); LeanSearch uses neural representations for semantic retrieval (Gao et al., 2024); LeanFinder adapts semantic retrieval to informal user intent through fine-tuning on synthesized queries (Lu et al., 2026); and LeanSearchV2 further combines semantic retrieval with LLM-based query expansion and reasoning-aware reranking (Gao et al., 2026). A complementary family conditions retrieval on the current proof state. ReProver introduced dense state–premise retrieval for Lean by encoding the goal and its local context (Yang et al., 2023). Subsequent work has extended this setting with multi-stage retrieval and cross-encoder reranking (Tao et al., 2025), or by incorporating premises identified through hammer-style automation (Zhu et al., 2026).

Recent work has also begun to exploit Mathlib's dependency structure, though in different ways. GNNReProver incorporates dependency-graph information into the representations of proof states

and premises through relational message passing (Petrovčič et al., 2025). It still ranks premises by vector matching in the resulting representation space, so the graph affects the matching scores but does not introduce new candidates beyond those considered by direct retrieval. LeanExplore computes a global PageRank score over the entire library and uses it as a structural ranking bias (Asher, 2025); this is a static, query-independent prior that reweights candidates without expanding the retrieval pool. GARNET differs from both: it treats the dependency graph not as an encoding signal or a global prior, but as a source of additional candidates. After retrieving an initial set A through direct matching, GARNET expands it to $A \cup \mathcal{N}(A)$ by following local dependency neighborhoods, then reranks the union using both retrieval and graph-derived features. In other words, GARNET uses the graph for post-retrieval candidate discovery, rather than only for representing or reweighting the original candidates.

2.2 Automatic Theorem Proving

LeanHammer (Zhu et al., 2026) retrieves premises conditioned on the proof state, then translates goals to external non-LLM provers such as Zipperposition (Vukmirović et al., 2022) and cvc5 (Barbosa et al., 2022) for small subgoals. Yang et al. (2023) jointly train a state-conditioned retriever and a tactic generation model. At each proof step, the retriever selects relevant premises, and the tactic model generates the next tactic based on the current goal and retrieved context.

Recent works use large language models as proof agents, mostly with query-based retrieval. COPRA (Thakur et al., 2024) retrieves external lemmas to prompt a general-purpose LLM within a backtracking search. Hilbert (Varambally et al., 2026) recursively decomposes goals and retrieves theorems semantically. LEAP (Kung et al., 2026) decomposes problems into informal blueprints refined against the Lean compiler. LeanMarathon (Zhang et al., 2026) coordinates multiple agents around a shared blueprint for long-horizon formalization.

3 METHOD

We describe GARNET in two parts: candidate construction from query- and state-conditioned retrieval, followed by graph-aware reranking of the constructed candidate pool.

3.1 Problem Setup and Candidate Construction

Given a natural-language proof-intent query q and a current proof state s, the goal is to rank useful declarations from a premise library $\mathcal{B}$. Candidates come from the released Mathlib premise bank of the benchmark's version; user-defined or other external declarations are out of scope. We use two frozen encoders, E_Q and E_S, to measure compatibility with the semantic query and the local proof context, respectively. The query encoder also provides representations for library declarations. For $c \in \mathcal{B}$, we define

$$\rho_Q(c) = \cos\big(E_Q(q), E_Q(c)\big), \qquad \rho_S(c) = \cos\big(E_S(s), E_S(c)\big), \tag{1}$$

where $\rho_Q(c)$ and $\rho_S(c)$ are the query–declaration and state–declaration similarity scores.

We retrieve declarations independently from the two views:

$$Q_K(q) = \mathrm{TopK}_{c\in\mathcal{B}}\, \rho_Q(c), \qquad S_K(s) = \mathrm{TopK}_{c\in\mathcal{B}}\, \rho_S(c), \tag{2}$$

with $K = 150$ per channel for the joint M11 configuration. Single-view inference uses a matched total anchor budget, as detailed in Appendix C. Their union forms the initial anchor set

$$A = Q_K(q) \cup S_K(s). \tag{3}$$

Using both channels allows the anchor set to retain proof intent relevance from q while also reflecting the local proof context encoded by s.

Dependency-guided candidate expansion. Direct retrieval provides precise anchors, but a required premise need not itself be highly similar to the query or the proof state. It may instead be connected to a retrieved anchor through the library's dependency structure. We therefore construct a directed uses graph from the Mathlib 4.28 corpus, where $u \to v$ indicates that declaration u uses declaration

v in its definition or proof. Formally, $G^{\to} = (\mathcal{B}, \mathcal{E}^{\to})$. Outgoing neighbors are referred to as *callees*, and incoming neighbors as *users*.

The graph has 310,579 declarations and approximately 6.3 million directed uses edges. Its two edge directions have markedly different degree profiles: the out-degree quartiles are $(6, 14, 26)$, whereas the in-degree quartiles are $(0, 1, 3)$, with maximum degrees of 448 and 63,635, respectively. Thus, the incoming degree distribution has a substantially heavier tail. Further graph statistics are reported in Appendix A.

These statistics motivate an asymmetric expansion budget. To avoid prioritizing broadly connected declarations, we first rank outgoing neighbors by increasing global in-degree and incoming neighbors by increasing global out-degree. We then retain the top-ranked neighbors under separate budgets:

$$\mathcal{D}^{(D)}(a) = \mathrm{Top}_{D}^{\uparrow d_{\mathrm{in}}}\{v : (a, v) \in \mathcal{E}^{\to}\}, \qquad \mathcal{R}^{(R)}(a) = \mathrm{Top}_{R}^{\uparrow d_{\mathrm{out}}}\{v : (v, a) \in \mathcal{E}^{\to}\}. \tag{4}$$

where d_{in} and d_{out} denote the global in-degree and out-degree, respectively, and $\mathrm{Top}_{m}^{\uparrow d}$ retains the first m elements after sorting by increasing d.

The expanded candidate pool is

$$C = A \cup \bigcup_{a \in A} \Big(\mathcal{D}^{(D)}(a) \cup \mathcal{R}^{(R)}(a) \Big), \tag{5}$$

where duplicate declarations are merged. This construction preserves the direct evidence provided by retrieval while allowing structurally related declarations to enter the reranking pool, even when they are not among the direct Top-K results.

3.2 View-Conditioned Graph Reranking

Given the fixed candidate pool C from Sec. 3.1, GARNET assigns every $c \in C$ a ranking score

$$S(c) = S_E(c) + g_G(c)\, \Delta_G(c), \tag{6}$$

where $S_E(c)$ is a direct-evidence score, $\Delta_G(c)$ is a graph residual, and $g_G(c) \in [0, 1]$ is a learned gate. Direct retrieval is the primary signal; graph context enters as a gated correction. The encoders E_Q and E_S remain frozen; only reranker parameters are learned.

Retrieval views. The reranker is designed to operate under varying input availability: the proving agent may propose only a natural-language query, only a proof-state identifier, or both. To accommodate these scenarios, the reranker is conditioned on which input channels are available. We write $\mathrm{has}_Q, \mathrm{has}_S \in \{0, 1\}$ for the query and state channels and consider three configurations: **M11** (joint; $\mathrm{has}_Q{=}\mathrm{has}_S{=}1$), **M10** (query-only), and **M01** (state-only). Let $h_q{=}W_q E_Q(q)$ and $h_s{=}W_s E_S(s)$. The context vector h_x is

$$h_x = \begin{cases} h_q, & \text{M10}, \\ h_s, & \text{M01}, \\ \alpha h_q + (1 - \alpha) h_s, & \text{M11}, \end{cases} \tag{7}$$

where, in the joint case,

$$\alpha = \sigma\Big(\mathrm{MLP}_{\mathrm{fuse}}\big[h_q,\, h_s,\, |h_q - h_s|,\, h_q \odot h_s\big] \Big). \tag{8}$$

Following Chen et al. (2017), we supply the fusion MLP with both the original view representations and their element-wise product and absolute difference to sharpen inference between the paired inputs. The utility of learning this fusion weight versus fixing it to a constant average is validated in Appendix G. We further define a view-dependent similarity used by both scoring branches,

$$\rho_{\mathrm{view}}(c) = \begin{cases} \rho_Q(c), & \text{M10}, \\ \rho_S(c), & \text{M01}, \\ \max\{\rho_Q(c), \rho_S(c)\}, & \text{M11}. \end{cases} \tag{9}$$

Direct-evidence branch. Each declaration $c \in C$ is projected as $h_c = W_c E_Q(c)$ and scored together with a compact feature vector $\phi_E(c) \in \mathbb{R}^8$ formed from retrieval similarities, ranks, and the

instance-level channel bits $(\mathrm{has}_Q, \mathrm{has}_S)$ (Appendix B). Similarly, we augment the direct-evidence MLP with the same element-wise product and absolute difference interactions between the context vector h_x and the candidate vector h_c:

$$S_E(c) = \mathrm{MLP}_E([h_x,\, h_c,\, h_x \odot h_c,\, |h_x - h_c|,\, \phi_E(c)])\,. \tag{10}$$

The effectiveness of these matching interaction terms is quantified in Appendix G. This score is defined for all of C, including graph-expanded candidates absent from $Q_K(q)$ and $S_K(s)$.

Graph residual branch. Given the fixed candidate pool C, the graph branch aggregates only the local structure induced by C. For each candidate $c \in C$, its in-pool neighborhood is

$$\mathcal{N}_C(c) = \{t \in C : (c,t) \in \mathcal{E}^{\rightarrow} \;\vee\; (t,c) \in \mathcal{E}^{\rightarrow}\}. \tag{11}$$

Graph aggregation therefore introduces no new candidates; it only exploits candidate–candidate usage edges within C. The direction of each edge is retained as a relation feature. The graph branch then proceeds in three steps: neighbor selection, neighbor encoding, and attention-based aggregation with gating.

Selection. From the in-pool neighborhood $\mathcal{N}_C(c)$, keep at most $P{=}24$ neighbors ordered by the active view,

$$\Pi_P(c) = \mathrm{TopP}_{t \in \mathcal{N}_C(c)}\; \rho_{\mathrm{view}}(t). \tag{12}$$

Encoding. For each $t \in \Pi_P(c)$,

$$z_t = \mathrm{MLP}_P([h_x,\, h_t,\, h_c,\, \phi_t(c)])\,, \qquad h_t = W_c E_Q(t), \tag{13}$$

where $\phi_t(c) \in \mathbb{R}^5$ summarizes the directed edge role, $\rho_{\mathrm{view}}(t)$, and hubness (Appendix B).

Attention and gating. Neighbors are aggregated by context-conditioned attention,

$$\beta_t = \frac{\exp(h_x^\top z_t)}{\sum_{u \in \Pi_P(c)} \exp(h_x^\top z_u)}, \qquad h_G(c) = \sum_{t \in \Pi_P(c)} \beta_t z_t, \tag{14}$$

and the residual is $\Delta_G(c) = w_G^\top h_G(c)$. A gate $g_G(c) = \sigma(\mathrm{MLP}_G[\cdots])$ (Appendix B) down-weights the graph term when local evidence is scarce. If $\Pi_P(c) = \varnothing$, we set $h_G(c) = 0$ and $g_G(c) = 0$, hence $S(c) = S_E(c)$. We ablate the contribution of this gated graph residual in Appendix G.

3.3 Training and Inference

Candidates are ranked by $S(c)$ in Eq. 6. Let $\mathcal{Y} \subseteq C$ be the gold premises present in C. The listwise loss is

$$\mathcal{L} = -\frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} \log \frac{\exp(S(y)/\tau)}{\sum_{c \in C} \exp(S(c)/\tau)}, \tag{15}$$

with temperature τ; instances with $\mathcal{Y} = \varnothing$ are omitted.

View dropout. During training, each instance is uniformly assigned to M11, M10, or M01. We keep the candidate pool fixed across views and apply dropout only to the reranker inputs: features from inactive channels are masked, and view-dependent scoring and neighbor selection use only the active inputs. This trains a single reranker to operate under different input availability. At inference, M11 uses both retrieval channels, whereas M10 and M01 use only the query and state channels, respectively; the latter therefore serve as inference-time view ablations of the same checkpoint. Further details are provided in Appendix C, with fully view-consistent retraining evaluated in Appendix G.

4 Experiments

4.1 Experimental Setup

We evaluate GARNET at two complementary levels, following the common practice of assessing both premise retrieval and downstream theorem proving (Yang et al., 2023; Gao et al., 2026). Premise-retrieval evaluation isolates whether the proposed retrieval architecture recovers relevant

library declarations, while end-to-end theorem proving tests whether these retrieval gains translate into successful proof construction when used by an LLM agent.

Retrieval Benchmarks. For retrieval, we construct an in-domain benchmark from Mathlib 4.28 using the LeanDojo pipeline (Yang et al., 2023), yielding 310,579 library premises and 479,801 tactic-level proving instances. We evaluate on two Mathlib-4.28 theorem splits: **Random** and **Novel-premise**. The latter contains test theorems that depend on at least one premise not used by any training theorem and preferentially includes test cases whose gold premises have low frequency in the Mathlib corpus. Partition sizes are reported in Appendix F.1. We additionally evaluate on Lean Workbook (Ying et al., 2025), a large-scale problem set formalized from natural-language math problems.

Theorem proving from proof sketches. For downstream evaluation, we curate 60 theorems from ProofNet-Verified (An et al., 2026)—which we call **SketchFill-60**—and derive proof sketches from their verified proofs by retaining the outermost `have` subgoals and replacing each subproof with a `sorry` placeholder. Selection criteria and sketch construction are detailed in Appendix I. This task reflects the standard mathematical practice of *subgoal decomposition*, which underpins interactive theorem proving (Yang et al., 2023; Varambally et al., 2026; Zhang et al., 2026). By preserving the human-written decomposition of a verified proof, the agent operates on well-defined subgoals while retrieving premises and constructing a proof for each of them. Importantly, each subgoal exposes a corresponding proof state, providing a common interface for comparing query-based, state-based, and joint query–state retrieval within the same LLM proving agent.

Encoders and baselines. The query encoder E_Q in GARNET is a frozen Qwen3-Embedding-8B. The state encoder E_S is initialized from the official LeanDojo ByT5 retriever checkpoint (Yang et al., 2023) and further fine-tuned for two epochs on the corresponding Mathlib 4.28 training split using the original LeanDojo training setup; this model also serves as the standalone **State-ByT5** baseline. GARNET reranker training details are provided in Appendix C. We compare against query-conditioned retrievers (LeanFinder and LeanSearchV2) and state-conditioned retrievers (LeanStateSearch, LeanPremise, and State-ByT5).

For LeanSearchV2, we use its retriever-only configuration, which uses the same Qwen3-Embedding-8B backbone as E_Q with direct cosine-similarity retrieval, and therefore serves as the direct query-retrieval baseline for GARNET. We exclude its optional reranking and LLM-based reasoning pipeline, as these form an orthogonal stage that can in principle be stacked on top of any first-stage retriever, including GARNET, while introducing additional per-query LLM inference cost.

We do not include GNNReProver (Petrovčič et al., 2025) as a quantitative baseline because we could not identify a released model checkpoint, while its reported system uses an ensemble of six independently trained GNN models on the original LeanDojo random split. Reproducing its reported configuration on our Mathlib 4.28 setup would therefore require retraining the full graph-model ensemble rather than evaluating a released retriever. Appendix F.3 provides version-alignment analyses for baselines trained on different Mathlib versions. Online per-call retrieval latency for all systems is reported in Appendix F.5.

End-to-end prover. We use Ax-Prover with GPT-5.6-Luna (Requena et al., 2026). Each proving attempt allows at most five agent iterations, and we run five independent attempts per theorem. Learned retrievers return Top-$K = 20$ premises per retrieval call. For GARNET, all interaction variants use the same checkpoint. **M10** conditions retrieval on a natural-language query, whereas **M11** conditions on both the query and the recorded proof state indexed by a provided subgoal identifier. **Adaptive** uses M11 when a subgoal identifier is provided and M10 otherwise. We additionally evaluate **M01** as a state-only ablation, conditioning retrieval only on the indexed proof state. Full interaction protocols and tool prompts are provided in Appendix I and Appendix J.

4.2 Evaluation Protocol and Metrics

External baselines use their released premise banks, checkpoints, and candidate-generation pipelines. GARNET uses split-specific fine-tuned state encoders. Following Gao et al. (2026), we report group Recall@k, the fraction of gold premises retrieved within the top k; Covered@k, the fraction of instances for which all gold premises are retrieved within the top k; and MRR, which

Table 1: End-to-end premise retrieval on Mathlib test sets. External baselines use their released premise banks, checkpoints, and candidate-generation pipelines. State-ByT5 and GARNET use the split-specific fine-tuned state encoders described in Section 4.1. The Input column indicates the retrieval input used by each system.

Split	System	Input	R@5	R@10	R@20	C@20	MRR
Random	LeanFinder (v4.28)	Query	0.203	0.251	0.296	0.231	0.196
	LeanSearchV2 (v4.28)	Query	0.193	0.246	0.300	0.232	0.182
	LeanStateSearch (v4.16)	State	0.149	0.210	0.272	0.212	0.130
	LeanPremise (v4.33)	State	0.121	0.182	0.259	0.194	0.118
	State-ByT5 (random-ft)	State	0.223	0.295	0.369	0.287	0.210
	GARNET (M10)	Query	0.312	0.394	0.483	0.377	0.306
	GARNET (M01)	State	0.303	0.395	0.489	0.388	0.290
	GARNET (M11)	**Query + State**	**0.405**	**0.509**	**0.607**	**0.492**	**0.383**
Novel	LeanFinder (v4.28)	Query	0.191	0.234	0.277	0.212	0.190
	LeanSearchV2 (v4.28)	Query	0.183	0.234	0.283	0.217	0.179
	LeanStateSearch (v4.16)	State	0.127	0.176	0.230	0.176	0.114
	LeanPremise (v4.33)	State	0.103	0.160	0.232	0.169	0.104
	State-ByT5 (novel-ft)	State	0.163	0.220	0.284	0.214	0.164
	GARNET (M10)	Query	0.259	0.336	0.420	0.316	0.259
	GARNET (M01)	State	0.215	0.292	0.376	0.281	0.219
	GARNET (M11)	**Query + State**	**0.326**	**0.422**	**0.521**	**0.404**	**0.319**

rewards ranking a relevant premise near the top. During training and testing on Mathlib data, we exclude the theorem being proved from the premise graph used for candidate expansion (Section 3.1), preventing information leakage.

For theorem proving, we report pass@k, the fraction of theorems solved within k independent attempts. We additionally report single-attempt accuracy as the mean $\pm$ standard deviation of the five attempt-wise solve rates over the same 60 theorems, summarizing average single-attempt performance and run-to-run variability. On successful traces, we additionally measure whether the premises eventually used in the accepted proof were surfaced by retrieval. $\mathrm{Cov}(\mathcal{R}, U)$ measures coverage of used premises across all retrieval calls in an attempt; Pair@K measures per-call recovery of used premises; and QHit@K measures the fraction of retrieval calls that return at least one eventually used premise. Formal definitions and aggregation details are provided in Appendix F.2.

4.3 Retriever Evaluation on Mathlib

Main Results. Table 1 reports end-to-end premise retrieval on the Mathlib random and novel-premise test sets. Systems are grouped by input modality: query-only retrievers (LeanFinder, LeanSearchV2), state-only retrievers (LeanStateSearch, LeanPremise, State-ByT5), and GARNET variants. GARNET (M11) is the full model with dual-input conditioning and graph reranking (Section 3). GARNET (M10) and GARNET (M01) are single-input ablations of the same trained checkpoint (Section 3.3).

On the random split, GARNET (M11) achieves R@20=60.7% and MRR=0.383, outperforming all baselines. The single-input variants also remain strong (M10 48.3%, M01 48.9%). On the novel-premise split, all systems degrade because the split constructively selects training-unseen/rare premises as gold targets, raising retrieval difficulty across all evaluated systems. GARNET (M11) still leads at R@20=52.1%. Here M10 (42.0%) outperforms M01 (37.6%), suggesting that when gold premises are distributionally rare, query-level semantic signals are more discriminative than local state signals.

Analysis of graph augmentation. We analyze the two graph-augmented stages: dependency-guided candidate expansion and graph-aware reranking. On the random validation split, one-hop expansion raises candidate-pool coverage from 82.5% to 93.2%, a $+1.7$-point gain over a matched query–state mixture without expansion (Appendix A.1). Removing expansion from the full model reduces R@30 and C@30 by 5.6 and 5.0 pp; with the pool fixed, removing the graph branch reduces

them by 2.6 and 2.4 pp (Appendix G). Removing both graph components reduces R@30 and C@30 by 7.5 and 6.7 pp, respectively. Together, dependency structure contributes at both stages: expansion recovers premises missed by direct retrieval, and graph-aware reranking improves their ranking.

4.4 Retriever Evaluation on Lean Workbook

We evaluate cross-dataset generalization on Lean Workbook (Ying et al., 2025), yielding 15,695 valid test cases. GARNET is trained only on the Mathlib random split.

Table 2: End-to-end premise retrieval on Lean Workbook ($n = 15{,}695$). The *Input* column indicates the retrieval input used by each system.

System	Input	R@5	R@10	R@20	C@20	MRR
LeanFinder (v4.28)	Query	0.060	0.086	0.134	0.121	0.055
LeanSearchV2 (v4.28)	Query	0.034	0.054	0.088	0.076	0.032
LeanStateSearch (v4.16)	State	0.077	0.116	0.159	0.150	0.059
LeanPremise (v4.33)	State	0.015	0.028	0.047	0.041	0.015
State-ByT5 (random-ft)	State	0.148	0.210	0.283	0.249	0.123
GARNET (M10)	Query	0.331	0.436	0.531	0.450	0.286
GARNET (M01)	State	0.304	0.414	0.527	0.442	0.274
GARNET (M11)	**Query + State**	**0.391**	**0.501**	**0.607**	**0.517**	**0.351**

GARNET (M11) achieves 60.7% R@20, compared with 28.3% for the strongest baseline, while M10 and M01 reach 53.1% and 52.7%. Among baselines, query-based retrieval degrades more from Mathlib to Lean Workbook than State-ByT5 (LeanFinder: 29.6→13.4%; LeanSearchV2: 30.0→8.8%; State-ByT5: 36.9→28.3%). M11 remains stronger than either single-input variant, further supporting the complementarity of query and state signals.

4.5 End-to-End Theorem Proving on SketchFill-60

Main results. GARNET (Adaptive) achieves the highest mean single-attempt accuracy among the evaluated systems, $77.7 \pm 3.0\%$, and also the highest pass@5 of 88.3% (Table 3). Paired theorem-level bootstrap confidence intervals for these comparisons are reported in Appendix H.1. The margin over state-only baselines is substantially larger than over query-only baselines: the latter reach 83–85% pass@5, whereas learned state-only baselines reach 65–73%. This state–query gap is notable in light of Lean Workbook, where state-based similarity degraded less than query-based matching. The reversal suggests that static retrieval quality alone does not determine agentic proving performance. In a dynamic proving loop, the LLM can repeatedly reformulate what mathematical fact it needs, whereas a state-conditioned interface ties retrieval to a predefined sketch subgoal and its recorded proof state.

Table 3: End-to-end proving on SketchFill-60. Single-attempt accuracy is reported as mean ± standard deviation across the five attempt-wise solve rates.

Method	pass@5	Single-attempt Acc. (%)
No Retrieval	63.3%	42.7 ± 2.5
LeanPremise	65.0%	53.3 ± 3.1
LeanStateSearch	65.0%	55.7 ± 2.5
State-ByT5	73.3%	57.7 ± 1.5
oracle-state	81.7%	70.7 ± 2.2
LeanFinder	85.0%	75.0 ± 2.0
LeanSearchV2	83.3%	74.0 ± 4.3
GARNET (M01)	70.0%	55.7 ± 1.9
GARNET (M10)	86.7%	72.7 ± 4.8
GARNET (M11)	81.7%	72.0 ± 1.4
GARNET (Adaptive)	**88.3%**	$\mathbf{77.7 \pm 3.0}$

To examine this interaction effect directly, we use the controlled GARNET variants defined in Section 4.1. Adding an explicit semantic query to state-conditioned retrieval (M01 → M11) improves both single-attempt accuracy and pass@5 (Table 3). This mirrors the offline retrieval results, where M11 consistently outperforms M01, suggesting that query semantics provide complementary guidance beyond the local proof state. Building on this complementarity, GARNET (Adaptive) allows the prover to freely choose between M10 and M11 according to its current retrieval need rather than committing to a fixed mode, and achieves the strongest overall performance. On first-success traces, it uses M10 for 37.0% and M11

for 63.0% of retrieval calls, confirming that the agent makes substantial use of both modes rather than collapsing to one (Appendix H).

We further use an oracle-state retriever to separate limitations of state retrieval quality from those of the subgoal-indexed interaction interface. Given a selected subgoal, the oracle returns the gold premises associated with its reference subproof. Its substantial improvement over the learned state-only systems is consistent with retrieval quality being one contributor, yet it still does not close the gap to the query-conditioned systems. Together with the controlled GARNET variants, this suggests that the state-only gap reflects both retrieval quality and the constraints of a subgoal-indexed interaction interface.

Retrieval quality beneath the surface. Pair@20 and QHit@20 are best interpreted within the same retrieval interface: state-based calls are tied to a specific subgoal, whereas query-based calls can be broader and reformulated across iterations. Under these matched comparisons, M10 improves over the query baselines on all three trace-level metrics, while M01 similarly improves over the learned state baselines. Looking across the whole successful proof, M11 raises used-premise coverage to 79.3%, compared with 56.4% for M01 and 64.0% for M10, showing that query and state provide complementary retrieval signals. Adaptive retains similarly high coverage (76.7%) while achieving the strongest end-to-end performance.

Table 4: Successful-trace retrieval quality on the 34 theorems solved by all systems. Metrics use each system's first successful attempt. All retrievers return $K = 20$ premises per call. *Solved* reports pass@5 on the full 60-theorem evaluation set.

System	Input	Solved	Pair@20	QHit@20	Cov$(\mathcal{R}, U)$
LeanFinder (v4.28)	Query	51/60	14.5%	51.5%	50.3%
LeanSearchV2 (v4.28)	Query	50/60	16.5%	59.0%	53.5%
LeanStateSearch (v4.16)	State	39/60	15.3%	56.6%	35.9%
LeanPremise (v4.33)	State	39/60	23.1%	67.2%	32.3%
State-ByT5 (random-ft)	State	44/60	17.4%	59.9%	51.6%
GARNET (M01)	State	42/60	**25.5%**	**80.7%**	56.4%
GARNET (M10)	Query	52/60	20.9%	69.9%	64.0%
GARNET (M11)	Query + State	49/60	22.6%	69.8%	**79.3%**
GARNET (Adaptive)	**Query + Optional state**	**53/60**	22.3%	69.3%	76.7%

5 Conclusion

We present GARNET, a graph-augmented reranking framework for Lean premise retrieval that combines query- and state-conditioned signals with local dependency structure. Our analyses show that graph augmentation contributes at both stages of retrieval: dependency-guided expansion recovers useful candidates missed by direct retrieval, while graph-aware reranking provides additional structural evidence for prioritizing them. Across retrieval evaluations derived from Mathlib 4.28 and Lean Workbook, GARNET improves R@20 over the strongest query- or state-only baseline by 23.7–32.4 absolute points.

When integrated into an LLM proving agent, GARNET (Adaptive) achieves $77.7 \pm 3.0\%$ mean single-attempt accuracy and 88.3% pass@5 on SketchFill-60. Controlled variants and successful-trace analyses show that query and proof-state signals provide complementary retrieval behavior: joint query–state conditioning substantially increases coverage of the premises used across a successful proof, while adaptive retrieval achieves the highest end-to-end performance among the evaluated systems.

Overall, these results suggest that agentic premise retrieval benefits from combining semantic intent, formal proof state, and library structure, alongside a flexible interface for query- and state-conditioned retrieval.

AI USE STATEMENT

Generative AI tools were used in two capacities. First, as components of the experimental pipeline, as disclosed in the corresponding methodology sections: DeepSeek-V4-Flash was used to generate semantic retrieval queries for the retrieval experiments, and the Ax-Prover agent based on GPT-5.6-Luna was used as the theorem-proving agent in our end-to-end experiments. Second, generative AI tools were used to assist with preparation of the paper, including language editing and improving clarity and readability. They also provided feedback on the presentation of the experimental methodology and protocol and on the interpretation and framing of experimental results. Generative AI tools also assisted with the preparation and visual polishing of figures. All AI-assisted text, figures, and suggestions were reviewed by the authors, and technical claims and experimental details were checked against the underlying results and implementation. The authors take full responsibility for the final content of this work.

ETHICS STATEMENT

This work focuses on retrieval methods for Lean theorem proving and does not involve human subjects or sensitive personal data. We do not identify any significant ethical concerns arising from the methodology or experiments presented in this work.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the experimental setup and evaluation protocols in the main text and appendix. The construction and statistics of the retrieval benchmarks are documented in Appendix F.1, following the premise-usage extraction procedure of LeanDojo (Yang et al., 2023). The semantic query generation procedure, including the LLM used for annotation, is described in Appendix D. We report the evaluation metrics for both retrieval and theorem proving, as well as the computational environment used for all experiments described in Appendix F. Reranker training and implementation settings are reported in Appendix C.

REFERENCES

One An, Marcus J Min, Xujie Si, and Osbert Bastani. Ground false: Dissecting errors in formal mathematics benchmarks. In *3rd AI for Math Workshop: Toward Self-Evolving Scientific Agents*, 2026.

Justin Asher. Leanexplore: A search engine for lean 4 declarations, 2025. URL https://arxiv.org/abs/2506.11085.

Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 415–442. Springer, 2022.

Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for natural language inference. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1657–1668, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1152. URL https://aclanthology.org/P17-1152/.

Guoxiong Gao, Haocheng Ju, Jiedong Jiang, Zihan Qin, and Bin Dong. A semantic search engine for mathlib4. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8001–8013, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.470. URL https://aclanthology.org/2024.findings-emnlp.470/.

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

Po-Nien Kung, Linfeng Song, Dawsen Hwang, Jinsung Yoon, Chun-Liang Li, Simone Severini, Mirek Olšák, Edward Lockhart, Quoc V Le, Burak Gokturk, Thang Luong, Tomas Pfister, and Nanyun Peng. Leap: Supercharging llms for formal mathematics with agentic frameworks, 2026. URL https://arxiv.org/abs/2606.03303.

Lean FRO. Loogle! Lean 4 library search. https://loogle.lean-lang.org/. Accessed: 2026-08-22.

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

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

Job Petrovčič, David Eliecer Narvaez Denis, and Ljupčo Todorovski. Combining textual and structural information for premise selection in lean, 2025. URL http://arxiv.org/abs/2510.23637. version: 2.

Borja Requena, Austin Letson, Krystian Nowakowski, Izan Beltran-Ferreiro, and Leopoldo Sarra. A minimal agent for automated theorem proving, 2026. URL https://arxiv.org/abs/2602.24273.

Yicheng Tao, Haotian Liu, Shanwen Wang, and Hongteng Xu. Learning an effective premise retrieval model for efficient mathematical formalization, 2025. URL https://arxiv.org/abs/2501.13959.

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. An in-context learning agent for formal theorem-proving. In *First Conference on Language Modeling*, 2024.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, pp. 367–381. ACM, 2020. doi: 10.1145/3372885.3373824.

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively building formal proofs with informal reasoning, 2026. URL https://arxiv.org/abs/2509.22819.

Petar Vukmirović, Alexander Bentkamp, Jasmin Blanchette, Simon Cruanes, Visa Nummelin, and Sophie Tourret. Making higher-order superposition work. *Journal of Automated Reasoning*, 66: 541–573, 2022.

Rongge Xu, Hui Dai, Yiming Fu, Jiedong Jiang, Tianjiao Nie, Junkai Wang, Holiverse Yang, and Zhi-Hao Zhang. Leancat: A benchmark suite for formal category theory in lean (part i: 1-categories), 2026. URL https://arxiv.org/abs/2512.24796.

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

Huaiyuan Ying, Zijian Wu, Yihan Geng, Zheng Yuan, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems, 2025. URL https://arxiv.org/abs/2406.03847.

Yuanhe Zhang, Yuekai Sun, Taiji Suzuki, Jason D. Lee, and Fanghui Liu. Leanmarathon: Toward reliable ai co-mathematicians through long-horizon lean autoformalization, 2026. URL https://arxiv.org/abs/2606.05400.

Thomas Zhu, Joshua Clune, Jeremy Avigad, Qiaochu Jiang, and Sean Welleck. Premise selection for a lean hammer. In *International Conference on Learning Representations*, volume 2026, pp. 127594–127607, 2026.

A Graph Construction and Statistics

The graph is constructed by projecting dependency records onto the $N = 310{,}579$ declarations in $\mathcal{B}$, removing self-loops and duplicate edges. The resulting graph contains $6{,}297{,}705$ unique directed uses edges. Its degree distribution is highly asymmetric: the out-degree has quartiles $(6, 14, 26)$, whereas the in-degree has quartiles $(0, 1, 3)$. The maximum out-degree and in-degree are 448 and $63{,}635$, respectively. In addition, 13.2% of declarations have zero out-degree and 45.9% have zero in-degree. These statistics motivate the asymmetric expansion budgets $D = 20$ and $R = 4$ mentioned in Section 3.1.

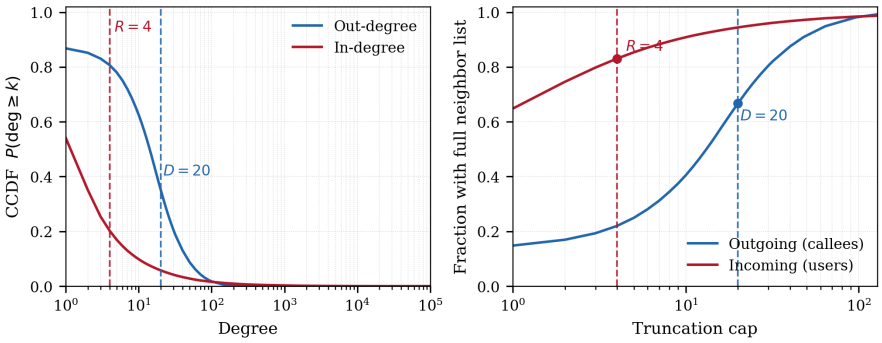


Figure 2: Degree distribution of the directed Mathlib uses graph. Out-degree (left) and in-degree (right) are shown on a logarithmic scale, illustrating the substantially heavier tail of incoming degrees. The dashed lines indicate the expansion budgets used by GARNET: $D = 20$ for outgoing callee edges and $R = 4$ for incoming user edges.

More than 60% of declarations have at most 20 outgoing edges and at most 4 incoming edges. Consequently, these budgets cover the complete one-hop neighborhoods of the majority of declarations. Since the expansion traverses only one graph hop, at most $D + R = 24$ neighbors are considered per anchor before deduplication. On the random validation split, under the default $\mathrm{Q}_{150} \cup \mathrm{S}_{150}$ configuration, the resulting expansion increases candidate-pool coverage from 82.52% for the initial anchor set to 93.16% after expansion; detailed results are reported in Appendix A.1.

A.1 Candidate-Pool Coverage

This analysis evaluates whether the candidate-construction procedure places required premises in the pool before ranking quality is measured. On the random validation split ($n = 47{,}410$), the default $\mathrm{Q}_{150} \cup \mathrm{S}_{150}$ anchor set covers 82.52% of gold groups. One-hop dependency expansion raises candidate-pool coverage to 93.16% (Table 5). The query and state channels provide complementary anchors: under the default configuration, 10.77% of gold groups are covered only by query retrieval, 24.21% only by state retrieval, and 47.54% by both. The table compares graph expansion against direct retrieval at matched candidate-pool sizes: query-only (73.94%), state-only (90.45%), and a query–state mixture that splits the budget evenly between the two channels (91.44%). Under the default configuration, expansion still retains a $+1.72$-point advantage over the matched mixture, separating the benefit of dependency structure from simply retrieving more candidates. Notably, for single-channel anchor sets the matched mixture also exceeds expansion alone (e.g., 91.65% vs. 86.97% for Q_{300}), further confirming that the two channels provide complementary anchors.

B Feature Specifications

Candidate features $\phi_E(c)$. In order: $\rho_Q(c)$, $\rho_S(c)$, $\rho_{\text{view}}(c)$, $\log(1 + \text{rank}(c; Q_K(q)))$, $\log(1 + \text{rank}(c; S_K(s)))$, $\log(1 + \text{rank}_{\text{view}}(c))$, has_Q, has_S. Ranks are 0 when c is absent from the corresponding Top-K list. The bits $(\text{has}_Q, \text{has}_S)$ are *instance-level* (identical for all $c \in C$) and encode the active view $\in \{(1, 1), (1, 0), (0, 1)\}$. On an inactive channel, that channel's ρ and $\log(1 + \text{rank})$ are set to 0. For ρ_{view} / $\text{rank}_{\text{view}}$: joint uses $\max(\rho_Q, \rho_S)$ and, when both ranks are positive, $\min(\text{rank}_Q, \text{rank}_S)$; a single-view collapses to the active channel.

Table 5: Candidate-pool coverage on the Mathlib random validation split ($n = 47{,}410$). $|A|$ and $|C|$ denote mean anchor-set and expanded candidate-pool sizes. Q-only$_{|C|}$, S-only$_{|C|}$, and QS-half$_{|C|}$ report direct retrieval coverage without graph expansion, using the same number of candidates as the expanded pool; QS-half splits the matched budget evenly between the query and state channels. Coverage values are percentages.

Method	$\lvert A\rvert$	A cov.	$\lvert C\rvert$	C cov.	Q-only$_{\lvert C\rvert}$	S-only$_{\lvert C\rvert}$	QS-half$_{\lvert C\rvert}$
$\mathbf{Q}_{300}$	300.0	63.60	1457.8	86.97	74.23	90.62	91.65
$\mathbf{S}_{300}$	300.0	78.34	1403.2	88.88	73.94	90.38	91.34
$\mathbf{Q}_{150} \cup \mathbf{S}_{150}$	**279.1**	**82.52**	**1372.1**	**93.16**	73.94	90.45	91.44

Neighbor features $\phi_t(c)$. In order: $\rho_{\text{view}}(t)$, directed role $r(c,t) \in \{+1,-1,0\}$, $\log(1+\text{hub}(c))$, $\log(1+\text{hub}(t))$, $\log(1+\text{rank}(t;\Pi_P(c)))$, where $\text{hub}(\cdot) = \text{outdeg} + \text{indeg}$ and $\text{rank}(t;\Pi_P(c))$ is the 1-based position of t in the view-ordered set $\Pi_P(c)$.

Gate inputs. The gate MLP consumes $h_G(c)$, $n_P(c) = |\Pi_P(c)|$, $\log(1+n_P(c))$, $m_{\text{sim}}(c) = \min_{t\in\Pi_P(c)} \rho_{\text{view}}(t)$, $m_{\text{hub}}(c) = \log(1+\text{hub}(c))$, and $\log(1+|\mathcal{N}_C(c)|)$. When $\Pi_P(c) = \varnothing$, set $h_G(c) = 0$ and $g_G(c) = 0$.

C Training and Testing Details

The GARNET reranker is trained with AdamW using the settings in Table 6. Training batches are bucketed by candidate-pool size, and the encoders E_Q and E_S remain frozen during reranker training.

Table 6: Training and implementation settings for the GARNET reranker.

Setting	Value
Optimizer	AdamW
Learning rate	5×10^{-4}
Weight decay	10^{-4}
Gradient clipping	2.0
Training epochs	3
Random seed	0
Listwise temperature τ	1.0
Projection width d	128
Dropout	0.1
Max. cases per batch	48
Candidate-pool budget per batch	49,152
Neighbor budget P	24
Validation metric	Recall@30 under M11
Validation cases	2,000
Checkpoint selection	Best validation checkpoint

View dropout during training. The ranker is trained with *view dropout*: each instance is randomly assigned one of three input views—M11 (both channels active), M10 (query only), or M01 (state only)—each with probability $1/3$. The anchor set A is always formed by retrieving $K = 150$ anchors per channel and taking their union, regardless of the sampled view. The candidate pool therefore remains fixed, while all view-conditioned reranker inputs and operations use only the active channels. This serves two purposes: (i) it regularizes the model by preventing over-reliance on any single modality, and (ii) it yields a single checkpoint that supports all three inference configurations without retraining. We compare this shared-pool training design with fully view-consistent single-view retraining in Appendix G.

Single-view inference. At test time, M11 uses the same $K = 150$ per-channel retrieval as in training. For M10 and M01, we instead retrieve $K = 300$ anchors from only the active channel to

form the anchor set, then apply the same graph expansion and set the corresponding view bits. The same trained checkpoint is used; thus, M10 and M01 are inference-time view ablations of the shared checkpoint rather than separately trained single-view models.

Candidate expansion retains up to $D = 20$ outgoing and $R = 4$ incoming neighbors per anchor. During training and testing on Mathlib data, we exclude the theorem being proved from the premise graph, ensuring that its own declaration is not available as a candidate premise. This prevents information leakage and simulates the realistic setting where the theorem to be proved is not yet part of the library.

D LLM Semantic Query Generation

LeanDojo (Yang et al., 2023) extracts proof states, tactics, and premise usage for each proof step. We extend this annotation by generating a natural-language retrieval query for each proof step using DeepSeek-V4-Flash.

Input context. For each proof step, the LLM receives: (1) the proof state immediately before the tactic application, and (2) the Lean proof code visible up to that point (truncated to the last 2,500 tokens).

Prompt design. The LLM is prompted with a system instruction followed by a user query containing the concrete proof state and prefix.

```
System: You assist Lean 4/mathlib premise retrieval.

Given the current Lean proof state and the visible proof prefix
before the next proof step, write exactly one concise semantic
search query describing what mathematical fact, transformation,
or relationship would likely be useful next.

A proof step may rely on one or more related library facts.
Express the overall local need as a single query rather
than multiple alternatives.

Do not output or guess a theorem or lemma name, Lean tactic code,
or a proof. You may reuse function, type, and operation names
already visible in the input when useful.

Use 10--30 words. Return only one sentence of plain text.

User: Current Lean proof state:
<state_before>

Proof prefix visible so far:
<proof_prefix>

Write the retrieval query.
```

Here, <state_before> and <proof_prefix> are replaced with the actual proof state and visible code prefix for each instance.

E Candidate Coverage and Entry Mix

Table 7 reports coverage across anchor budgets and query/state mixtures on the Mathlib random validation split ($n = 47{,}410$). We use a total anchor budget of 300 with the balanced $Q_{150} \cup S_{150}$ configuration as a coverage–efficiency trade-off. Increasing the budget to 400 improves candidate-pool coverage by at most about one percentage point while increasing the expanded pool from

approximately 1.37K to 1.73K candidates. Within budget 300, shifting additional anchors toward the state channel yields only marginal coverage gains, so we retain the balanced configuration.

Table 7: Anchor and expanded candidate-pool coverage on the Mathlib random validation split ($n = 47{,}410$). Coverage values are percentages.

Budget	Mix	mean $\lvert A\rvert$	A cov.	mean $\lvert C\rvert$	C cov.
200	Q_{200}	200.0	60.59	1044.6	84.94
200	S_{200}	200.0	74.60	1008.7	86.39
200	$Q_{150} \cup S_{50}$	189.5	76.77	1007.7	90.74
200	$Q_{100} \cup S_{100}$	186.9	79.48	991.2	91.61
200	$Q_{50} \cup S_{150}$	189.6	80.36	989.8	91.63
300	Q_{300}	300.0	63.60	1457.8	86.97
300	S_{300}	300.0	78.34	1403.2	88.88
300	$Q_{200} \cup S_{100}$	281.0	81.10	1386.4	92.81
300	$\mathbf{Q_{150} \cup S_{150}}$	**279.1**	**82.52**	**1372.1**	**93.16**
300	$Q_{100} \cup S_{200}$	280.9	83.27	1367.8	93.27
400	Q_{400}	400.0	65.69	1847.5	88.24
400	S_{400}	400.0	80.85	1774.2	90.63
400	$Q_{300} \cup S_{100}$	376.9	82.10	1765.0	93.43
400	$Q_{200} \cup S_{200}$	370.8	84.48	1728.5	94.10
400	$Q_{100} \cup S_{300}$	376.7	85.37	1726.0	94.21

F Experimental Details and Additional Results

All local retrieval and reranker experiments were conducted on a single NVIDIA GeForce RTX 4080 GPU with approximately 62 GB of system memory.

F.1 Dataset Statistics

Table 8 reports partition sizes before any filtering.

Table 8: Dataset partition sizes.

Dataset	Partition	Instances
Mathlib–Random	Train	384,059
	Validation	47,410
	Test	48,332
Mathlib–Novel	Train	351,443
	Validation	63,454
	Test	64,904
Lean Workbook	Evaluation	15,695

F.2 Evaluation Metrics

Offline premise retrieval. Gold premises are organized into *groups*, where each group $g \in \mathcal{G}$ contains a set of declaration identifiers $\mathrm{Docs}(g)$ that are interchangeable for a given proof step (e.g., alternative formulations of the same lemma). In our annotation, every group is a singleton: each gold declaration forms its own group with no alternatives.

A group is *hit* at rank k iff $\mathrm{TopK} \cap \mathrm{Docs}(g) \neq \varnothing$.

Group Recall@k is the fraction of gold groups hit:

$$\mathrm{R@}\,k = \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} \mathbb{1}[\mathrm{TopK} \cap \mathrm{Docs}(g) \neq \varnothing]\,. \tag{16}$$

Covered@k is the fraction of instances $i \in \mathcal{T}$ for which *every* gold group is hit:

$$\text{Covered}\,@k = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \mathbb{1}\Big[\bigwedge_{g \in \mathcal{G}_i} \big(\text{TopK}_i \cap \text{Docs}(g) \neq \varnothing\big)\Big]. \tag{17}$$

Under our singleton-group annotation (one group per gold declaration), this is equivalent to $\text{Gold}(i) \subseteq \text{TopK}_i$.

MRR is the mean reciprocal rank of the first hit in each group's ranking.

End-to-end proving metrics. Let $y_{ir} \in \{0, 1\}$ indicate whether theorem i is solved on independent attempt r, with $N = 60$ theorems and $R = 5$ attempts. The solve rate of attempt r is

$$A_r = \frac{1}{N} \sum_{i=1}^{N} y_{ir}.$$

We report the single-attempt accuracy as the mean and sample standard deviation across the five attempt-wise solve rates,

$$\bar{A} = \frac{1}{R} \sum_{r=1}^{R} A_r, \qquad s_A = \sqrt{\frac{1}{R-1} \sum_{r=1}^{R} (A_r - \bar{A})^2}.$$

The mean summarizes average single-attempt proving performance, while s_A quantifies variability across independent attempts; lower standard deviation therefore indicates greater run-to-run stability. This statistic is distinct from pass@k: pass@1 uses the first attempt, whereas $\bar{A}$ averages the solve rates of all five independent attempts.

Online retrieval on successful proofs. Metrics below are computed on a *common solved subset*: we restrict to the theorems solved by every retrieval system under comparison (34 of 60 for the nine systems in Table 4) and use each system's *first successful attempt* on each such theorem. Fixing the theorem set isolates retrieval behavior from differences in which theorems each system happens to solve, and gives every system equal weight in the aggregation. Name matching is exact Mathlib declaration identity (or shared Lean suffix). Proof-side identifiers that are not Mathlib declarations are dropped from U.

Let π be a successful approved proof, $U(\pi)$ the set of Mathlib premises used in π, $N(\pi)$ the number of `lean_search` calls on that attempt, and R_j the Top-K names returned by the j-th call. Write $\mathcal{R}(\pi) = \bigcup_{j=1}^{N(\pi)} R_j$. Then

$$\text{Cov}(\mathcal{R}, U) = \frac{\sum_\pi |\mathcal{R}(\pi) \cap U(\pi)|}{\sum_\pi |U(\pi)|}, \tag{18}$$

$$\text{Pair@}K = \frac{\sum_\pi \sum_{j=1}^{N(\pi)} \sum_{u \in U(\pi)} \mathbb{1}\{u \in R_j\}}{\sum_\pi N(\pi)\,|U(\pi)|}, \tag{19}$$

$$\text{QHit@}K = \frac{\sum_\pi \sum_{j=1}^{N(\pi)} \mathbb{1}\{\exists u \in U(\pi) :\, u \in R_j\}}{\sum_\pi N(\pi)}. \tag{20}$$

Thus $\text{Cov}(\mathcal{R}, U)$ measures *union* coverage of used premises across the whole attempt; Pair@K and QHit@K measure *per-call* hit rates.

F.3 Pairwise Coverage-Aligned Comparisons

LeanStateSearch (v4.16) and LeanPremise (v4.33) use premise libraries extracted from different Mathlib versions than our benchmark (4.28). To isolate ranking quality from premise-availability differences, we compute pairwise *coverage-aligned* subsets: for a given baseline m, we retain only test instances where every gold premise group is represented in both $\mathcal{B}_{\text{GARNET}}$ and $\mathcal{B}_m$. Metrics are then computed from each system's own ranking on this subset.

Table 9: Pairwise coverage-aligned comparison with LeanStateSearch (v4.16). Only instances with all gold groups present in both premise banks are retained. Retained (%) shows the fraction of the original split kept after alignment.

Split	System	Retained (%)	R@5	R@10	R@20	C@20	MRR
Random	LeanStateSearch (v4.16)	57.7	0.236	0.329	0.423	0.368	0.182
	GARNET (M11)		**0.478**	**0.580**	**0.669**	**0.605**	**0.406**
Novel	LeanStateSearch (v4.16)	52.5	0.217	0.299	0.386	0.336	0.167
	GARNET (M11)		**0.399**	**0.495**	**0.588**	**0.527**	**0.336**
Workbook	LeanStateSearch (v4.16)	69.7	0.108	0.163	0.222	0.216	0.080
	GARNET (M11)		**0.428**	**0.535**	**0.638**	**0.599**	**0.346**

LeanStateSearch (v4.16). Table 9 reports the GARNET–LeanStateSearch comparison. The aligned subset retains 57.7% of random-test instances, 52.5% of novel-test instances, and 69.7% of workbook instances.

LeanPremise (v4.33). Table 10 reports the GARNET–LeanPremise comparison. The aligned subset retains 97.3% of random-test instances, 97.7% of novel-test instances, and 97.5% of workbook instances.

Table 10: Pairwise coverage-aligned comparison with LeanPremise (v4.33). Only instances with all gold groups present in both premise banks are retained. Retained (%) shows the fraction of the original split kept after alignment.

Split	System	Retained (%)	R@5	R@10	R@20	C@20	MRR
Random	LeanPremise (v4.33)	97.3	0.123	0.186	0.264	0.199	0.119
	GARNET (M11)		**0.411**	**0.515**	**0.614**	**0.500**	**0.385**
Novel	LeanPremise (v4.33)	97.7	0.104	0.163	0.236	0.173	0.105
	GARNET (M11)		**0.330**	**0.426**	**0.526**	**0.411**	**0.321**
Workbook	LeanPremise (v4.33)	97.5	0.015	0.029	0.048	0.042	0.015
	GARNET (M11)		**0.398**	**0.510**	**0.618**	**0.529**	**0.355**

These pairwise tables isolate ranking differences by conditioning on common gold availability. They complement, rather than replace, the end-to-end results in the main text.

F.4 State-Encoder Fine-Tuning

Table 11: Original vs. fine-tuned ByT5 state encoders.

		@5		**@10**		**@20**		**@30**		**@50**		**MRR**
Split	**Model**	R	C	R	C	R	C	R	C	R	C	
Random	Official checkpoint	0.217	0.165	0.275	0.210	0.327	0.253	0.357	0.277	0.396	0.309	0.216
	Random-ft	0.223	0.172	0.295	0.228	0.369	0.287	0.415	0.324	0.474	0.374	0.210
Novel	Official checkpoint	0.169	0.125	0.216	0.161	0.264	0.197	0.291	0.218	0.326	0.246	0.175
	Novel-ft	0.163	0.122	0.220	0.165	0.284	0.214	0.324	0.245	0.380	0.289	0.164

Table 11 compares the original LeanDojo ByT5 checkpoint with our split-specific fine-tuned models. Fine-tuning uses only the corresponding training partition; no test data are accessed.

F.5 Online Retrieval Latency

We measure per-call online retrieval latency on the same hardware used for the local retrieval experiments. For each system, we time 24 calls after one untimed warm-up call. Query and proof-state representations are encoded on demand, without precomputed input embeddings or cached retrieval

results. Static premise embeddings are kept resident on GPU where applicable, reflecting their reuse across online retrieval calls. GARNET timings include input encoding, graph-based candidate expansion, and reranking. Query inputs are drawn from Adaptive first-success traces, with state inputs taken from the corresponding recorded SketchFill subgoals.

Table 12: Per-call online retrieval latency ($n = 24$ calls per system; $K = 20$).

System	Input	Mean (ms)	Median (ms)	p90 (ms)
LeanFinder (v4.28)	Query	173	166	178
LeanSearchV2 (v4.28)	Query	247	240	257
LeanStateSearch (v4.16)	State	20	20	22
LeanPremise (v4.33)	State	44	40	52
State-ByT5 (random-ft)	State	22	20	22
GARNET (M01)	State	190	188	200
GARNET (M10)	Query	223	222	230
GARNET (M11)	Query + State	246	242	256

GARNET requires 190–246 ms per retrieval call across its three fixed modes. M10 and M11 have latency comparable to the query-based retrievers, while the state-only baselines are substantially faster. Within GARNET, input encoding accounts for approximately 14–60 ms, candidate expansion and graph preparation for 160–180 ms, and final reranking for 8–12 ms. Adaptive selects M10 or M11 independently on each call and therefore incurs the latency of the selected mode.

G Ablation Study

To isolate the contribution of individual design choices, we evaluate model variants on a namespace-matched subset of the Mathlib random split. We restrict to theorems whose file namespace lies in Analysis ∪ Algebra ∪ RingTheory, the three largest namespaces in the split. This yields 131,158 training cases, 500 validation cases used for checkpoint selection, and 16,329 test cases. All models are trained for three epochs. Unless otherwise stated, we report M11 retrieval with both query and state available. Primary metrics are group Recall@k, Covered@k, and MRR.

Reranker ablations. We ablate the graph branch, learned query–state fusion, and matching interaction features while keeping the standard expanded candidate pool fixed.

Table 13: Reranker ablations on the namespace-matched test set ($n = 16{,}329$).

Model	R@5	R@30	C@5	C@30	MRR
Full	0.440	0.708	0.354	0.583	0.416
w/o graph branch	0.419	0.682	0.336	0.559	0.397
Fixed fusion $\alpha = \frac{1}{2}$	0.433	0.702	0.348	0.578	0.409
w/o interaction terms in S_E	0.422	0.690	0.338	0.566	0.400

Among the reranker ablations, removing the graph branch produces the largest drop, reducing R@30 from 0.708 to 0.682 (-2.6 pp) and C@30 from 0.583 to 0.559 (-2.4 pp). Removing the interaction terms also consistently hurts performance (R@30: -1.8 pp; C@30: -1.7 pp), while fixing the fusion weight has a smaller effect (R@30: -0.6 pp; C@30: -0.5 pp).

Candidate-pool expansion. We separately ablate the dependency-guided candidate expansion. The standard setting expands the $Q_{150} \cup S_{150}$ anchor set with one-hop dependency neighbors ($D = 20$, $R = 4$), whereas the no-expansion setting ranks only the initial anchors.

Removing candidate expansion substantially reduces retrieval performance: for the full model, R@30 drops from 0.708 to 0.652 (-5.6 pp) and C@30 from 0.583 to 0.533 (-5.0 pp). The same pattern holds without the graph branch (R@30: -4.9 pp; C@30: -4.3 pp). Conversely, the graph branch improves performance under both pool constructions, showing that candidate expansion and graph-aware reranking provide distinct benefits.

Table 14: Effect of dependency-guided candidate expansion on the namespace-matched test set ($n = 16{,}329$).

Pool	Model	R@5	R@30	C@5	C@30	MRR
Expanded	Full	0.440	0.708	0.354	0.583	0.416
Expanded	w/o graph branch	0.419	0.682	0.336	0.559	0.397
No expansion	Full	0.423	0.652	0.339	0.533	0.398
No expansion	w/o graph branch	0.406	0.633	0.326	0.516	0.385

Shared-pool versus view-consistent training. Our main checkpoint uses shared-pool view dropout: candidate construction always starts from $Q_{150} \cup S_{150}$, while the sampled view controls only the reranker inputs. Thus, M10 and M01 in the main experiments are obtained by changing the active view at inference. To test whether this training design materially affects single-view performance, we additionally train view-consistent variants from scratch on the same namespace-matched subset. For M10, both training and inference use candidates constructed from Q_{300}; for M01, both use S_{300}. All other training settings are unchanged, with checkpoint selection based on Recall@30 under the corresponding active view.

Table 15: Shared-pool view dropout versus fully view-consistent single-view training on the namespace-matched test set ($n = 16{,}329$). Shared-pool models use the same checkpoint trained from $Q_{150} \cup S_{150}$ candidates, whereas view-consistent models are retrained from scratch using only the active retrieval channel.

View	Training pool	R@5	R@30	C@5	C@30	MRR
M01	Shared $Q_{150} \cup S_{150}$	0.341	0.607	0.271	0.491	0.321
M01	View-consistent S_{300}	0.347	0.598	0.275	0.486	0.323
M10	Shared $Q_{150} \cup S_{150}$	0.358	0.626	0.283	0.504	0.346
M10	View-consistent Q_{300}	0.358	0.596	0.283	0.475	0.346

Table 15 shows similar early-ranking performance under the two training strategies.

H SketchFill-60 End-to-End Proving Details

Table 16 reports pass@k on SketchFill-60 using GPT-5.6-Luna as the proving agent. The oracle retriever (diagnostic only) returns exact gold premises associated with the selected subgoal.

Retrieval usage on successful traces. For each theorem solved within five attempts, we analyze its first successful attempt. Table 17 reports the mean number of retrieval calls per solved theorem. For the three query-conditioned GARNET interaction modes, we additionally decompose these calls according to whether retrieval uses the M10 or M11 view. GARNET (Adaptive) uses both views substantially: 37.0% of its retrieval calls use M10 and 63.0% use M11, showing that adaptive routing does not collapse to a single view.

H.1 Uncertainty Analysis for End-to-End Proving

To quantify uncertainty in the SketchFill-60 results, we perform a theorem-level paired bootstrap over the same 60 theorems used for all systems. We resample theorems with replacement for $B = 20{,}000$ replicates and report percentile 95% confidence intervals for the performance difference $\Delta = \text{Adaptive} - \text{Baseline}$. We consider both pass@5 and mean single-attempt accuracy over the five independent attempts.

Adaptive attains the highest point estimate on both metrics. The paired intervals confirm substantial improvements over the learned state-only systems, whereas the smaller differences relative to the strongest query-conditioned systems are not consistently resolved at the scale of SketchFill-60.

Table 16: End-to-end proof completion on SketchFill-60 ($n = 60$). Pass@k is the fraction of theorems proved within k attempts. Single-attempt accuracy is reported as mean $\pm$ standard deviation across the five independent attempt-wise solve rates.

System	Input	pass@1	pass@2	pass@3	pass@4	pass@5	Single-attempt Acc. (%)
No Retrieval	—	40.0%	48.3%	53.3%	61.7%	63.3%	42.7 ± 2.5
LeanPremise	State	50.0%	56.7%	60.0%	63.3%	65.0%	53.3 ± 3.1
LeanStateSearch	State	53.3%	60.0%	63.3%	65.0%	65.0%	55.7 ± 2.5
State-ByT5 (random-ft)	State	58.3%	68.3%	70.0%	71.7%	73.3%	57.7 ± 1.5
GARNET (M01)	State	58.3%	66.7%	66.7%	68.3%	70.0%	55.7 ± 1.9
oracle-state	State	73.3%	76.7%	76.7%	80.0%	81.7%	70.7 ± 2.2
LeanFinder	Query	75.0%	81.7%	81.7%	83.3%	85.0%	75.0 ± 2.0
LeanSearchV2	Query	68.3%	81.7%	81.7%	83.3%	83.3%	74.0 ± 4.3
GARNET (M10)	Query	68.3%	78.3%	81.7%	86.7%	86.7%	72.7 ± 4.8
GARNET (M11)	Query + State	71.7%	78.3%	80.0%	81.7%	81.7%	72.0 ± 1.4
GARNET (Adaptive)	**Query + Optional state**	**78.3%**	**83.3%**	**86.7%**	**88.3%**	**88.3%**	$\mathbf{77.7 \pm 3.0}$

Table 17: Retrieval usage on first-success traces of SketchFill-60. For each solved theorem, statistics are computed on its first successful attempt. "Calls/solved" is the mean number of retrieval calls per solved theorem. The lower panel decomposes these calls by GARNET view; percentages denote the share of retrieval calls using each view.

System	Solved	Calls/solved
No Retrieval	38	0.0
LeanPremise	39	14.2
LeanStateSearch	39	14.1
State-ByT5 (random-ft)	44	14.2
GARNET (M01)	42	8.0
LeanFinder	51	6.7
LeanSearchV2	50	6.5
GARNET (M10)	52	5.7
GARNET (M11)	49	12.2
GARNET (Adaptive)	**53**	**10.2**

GARNET mode	M10 calls/solved	M11 calls/solved
M10	5.7 (100%)	0
M11	0	12.2 (100%)
Adaptive	**3.8 (37.0%)**	**6.4 (63.0%)**

I SketchFill Protocol and Agent–Retriever Interaction

We construct the SketchFill evaluation set from verified gold proofs in ProofNet-Verified (An et al., 2026). To focus the evaluation on proofs that require non-trivial premise retrieval across multiple subgoals, we retain proofs with 3–14 outermost `have` subgoals, at least 8 distinct Mathlib declarations used in the gold proof, and non-trivial subproof bodies. Among the remaining candidates, we select the 60 theorems with the highest gold-premise diversity. We refer to this evaluation set as SketchFill-60.

Subset composition. Table 19 compares SketchFill-60 with the full ProofNet-Verified collection. Since ProofNet-Verified provides textbook metadata but no unified subject labels, we report the distribution across source textbooks together with formal- and informal-proof lengths. The selected subset remains distributed across the major textbooks while favoring proofs with the richer structure required by our SketchFill evaluation.

Table 18: Paired bootstrap 95% confidence intervals for the performance difference between GARNET (Adaptive) and all other systems on SketchFill-60. All differences are reported in percentage points.

	pass@5		Single-attempt accuracy	
Baseline	Δ	95% CI	Δ	95% CI
No Retrieval	+25.0	[+13.3, +36.7]	+35.0	[+25.3, +45.0]
LeanPremise	+23.3	[+13.3, +35.0]	+24.3	[+15.3, +34.0]
LeanStateSearch	+23.3	[+13.3, +35.0]	+22.0	[+12.7, +32.0]
State-ByT5	+15.0	[+6.7, +25.0]	+20.0	[+11.3, +29.0]
GARNET (M01)	+18.3	[+8.3, +28.3]	+22.0	[+13.0, +31.3]
oracle-state	+6.7	[0.0, +15.0]	+7.0	[0.0, +14.3]
LeanFinder	+3.3	[−3.3, +10.0]	+2.7	[−2.0, +7.7]
LeanSearchV2	+5.0	[0.0, +11.7]	+3.7	[−1.0, +8.3]
GARNET (M10)	+1.7	[0.0, +5.0]	+5.0	[+1.0, +9.3]
GARNET (M11)	+6.7	[+1.7, +13.3]	+5.7	[0.0, +11.7]

Table 19: SketchFill-60 compared with the full ProofNet-Verified set. Counts denote numbers of problems; proof lengths are medians. Formal length counts nonempty, non-comment lines in the gold Lean proof body, and informal length counts words in the natural-language proof.

	Count		Formal lines		Informal words	
Textbook	Full	SF-60	Full	SF-60	Full	SF-60
Dummit–Foote	87	8	56	78	83	117
Munkres	60	9	59	89	101	123
Rudin	56	10	56	70	64.5	62
Herstein	47	9	70	101	142	132
Artin	28	8	55	64	50	36
Axler	26	6	77	128	114	109
Ireland–Rosen	21	8	83	52	92	76
Shakarchi	16	1	181	34	90	31
Pugh	14	1	67	61	79	353
Putnam	12	0	294	—	176	—
All	367	60	67	67.5	91	94.5

Figure 3 illustrates how the LLM proving agent interfaces with retrievers under the SketchFill protocol. Starting from a ground-truth proof, we derive a sketch by preserving the outermost `have` subgoal structure and replacing each subproof body with a `sorry` placeholder tagged with a unique identifier (`-- [sf/hX]`). Each placeholder exposes a well-defined proof state that can be indexed by state-conditioned retrievers.

Query-only interaction. The agent supplies a natural-language description of the mathematical fact needed, and retrieval conditions only on this query. This interface is used by LeanFinder, LeanSearchV2, and GARNET M10.

State-only interaction. The agent supplies a subgoal identifier `hx` (e.g., `h0`), which indexes the recorded tactic state at that subgoal. Retrieval conditions only on this proof state. This interface is used by LeanPremise, LeanStateSearch, State-ByT5, and the GARNET M01 ablation.

Joint query–state interaction (GARNET M11). The agent must supply both a natural-language query and a subgoal identifier `hx` on every retrieval call. GARNET conditions on the query together with the recorded proof state of that subgoal and always uses the M11 view; query-only fallback is disabled.

Adaptive interaction (GARNET Adaptive). The agent always supplies a natural-language query and may optionally provide a subgoal identifier `hx`. Calls with `hx` use M11 (query + state), whereas calls without `hx` use M10 (query only). Thus, the retrieval view is selected independently on each call by the agent.

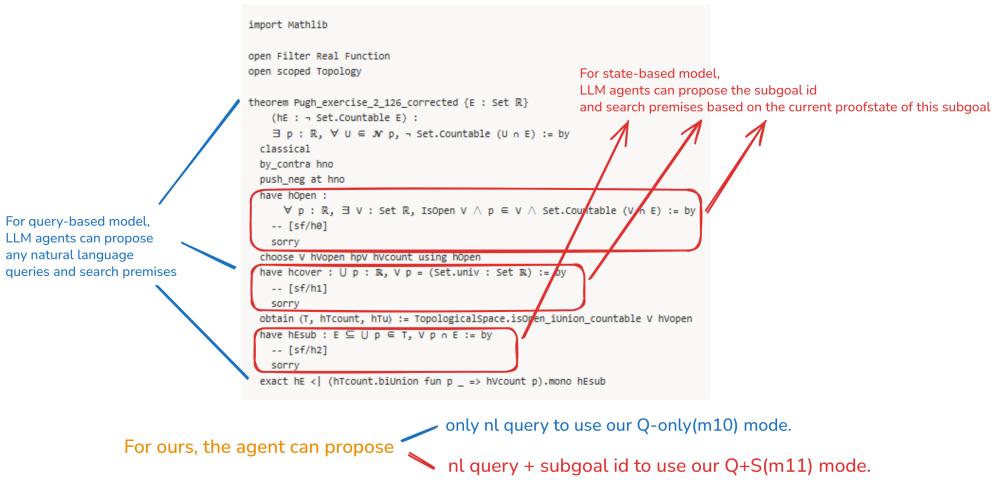


Figure 3: Agent–retriever interaction under the SketchFill protocol. Query-only systems condition on an NL retrieval query; state-only systems, including GARNET M01, condition on the recorded proof state indexed by `hx`. GARNET M11 requires both inputs on every call, whereas GARNET Adaptive uses M11 when `hx` is supplied and M10 otherwise.

J Tool Prompts and Oracle Retriever

J.1 Query-Only NL — LeanFinder / LeanSearchV2/ GARNET M10

```
Search Mathlib for lemmas that match a mathematical *need*.

Write the query as one concise natural-language sentence (about 10-30
words) describing the fact, rewrite, inequality, or construction that
would help the *current* proof state / next proof step. Prefer
describing the math in words so the search backend can map your need
to concrete declarations.

Good examples:
- "product of two ideals equals their intersection when the ideals
  sum to the whole ring"
- "finite group of prime-square order is abelian / commutative"
- "inducing maps compose for successive subspace topologies"
- "Cauchy-Schwarz on a finite weighted real sum"
- "nonconstant holomorphic map on a connected open set is open"
```

J.2 GARNET M11

```
Search Mathlib with GR-M11 (query + subgoal proof state).

Required:

- query: one concise natural-language sentence (10--30 words)
  describing the mathematical fact needed.
- hx: sketchfill subgoal id from comments like `-- [sf/h0]`
  (pass `h0`, `h3`, ...).
  Retrieval always uses GR-M11 with that subgoal's recorded proof state.
  Do not omit hx; query-only (M10) is disabled in this protocol.

When filling a `have ... := by sorry` marked `-- [sf/hX]`,
always pass that hx.
```

```
Examples:

- query="cardinality of this finite field is odd", hx="h0"
- query="X^n + constant is monic", hx="h2"
- query="finite-field multiplicative order of a nonzero element",
  hx="h10"
```

J.3 GARNET ADAPTIVE M10/M11 ROUTER

```
Search Mathlib for lemmas (hybrid GR: M10 vs M11).

Required:
- query: one concise natural-language sentence (10--30 words) describing
  the mathematical fact needed.

Optional:
- hx: sketchfill subgoal id from '-- [sf/h0]' (pass 'h0', 'h3', ...).
  - Omit hx --> GR-M10 (query-only): best for *general / global*
    Mathlib facts that do not depend on the local hypotheses of one
    subgoal (standard rewrites, algebra identities, cardinality/order
    facts, topology lemmas stated in ordinary mathematical language).
  - Pass hx --> GR-M11 (query + that subgoal's recorded proof state):
    use only when the need is *tied to that specific subgoal* --
    local hyps, a concrete goal shape at '-- [sf/hX]', or a lemma
    that must match that subgoal's context.

Examples:
- query="product of two ideals equals their intersection when the
  ideals sum to the ring"
- query="finite group of prime-square order is abelian"
- query="negation of both sides of an equality in a ring"
- query="Cauchy--Schwarz on a finite weighted real sum"
- query="this element has odd multiplicative order in the residue
  field", hx="h0"
- query="the induced map on this successive subspace topology is
  continuous", hx="h3"
```

J.4 STATE-ONLY HX — LEANPREMISE / LEANSTATESEARCH / STATE-BYT5 / GARNET M01

```
Search Mathlib using the Lean proof state of a sketchfill subgoal.

Required:
- hx: subgoal id from '-- [sf/hX]' (pass 'h0', 'h3', ...). Retrieval
  encodes that subgoal's recorded proof state (the NL query text is
  ignored by the backend).
- query: short placeholder (you may repeat the goal in words);
  backend does not use it.

Tool results are labeled with the subgoal id, e.g.
'=== hx=h0 | query=... (N matches) ===', so you can match each result
block to the corresponding sketchfill subgoal.

When filling a 'have ... := by sorry' marked '-- [sf/hX]', always pass
that hx.

Examples:
- query="odd cardinality", hx="h0"
```

```
- query="monic polynomial", hx="h2"
- query="char not two", hx="h1"
```

J.5 ORACLE STATE RETRIEVER

The oracle retriever is a **diagnostic** tool for separating state-retrieval quality from the subgoal-indexed interaction protocol. It exposes the same agent-visible interface as the real state-conditioned retrievers (Appendix J.4): the agent passes a subgoal index `hx`, and the backend returns the gold premises used in that subgoal's reference proof, formatted identically to ordinary search hits so the agent cannot distinguish oracle output from real retrieval.

Unlike the learned retrievers, which return at most 20 premises per call, the oracle returns up to 48 gold premises. We use this larger output cap to avoid truncating the reference premise set for subgoals whose gold subproof contains more than 20 declarations. The oracle is therefore used only as a diagnostic of the state-conditioned interaction protocol, rather than as a budget-matched retrieval baseline.