Lean Finder: Semantic Search for Mathlib That Understands User Intents

Jialin Lu [1] **Kye Emond** [2 3] **Weiran Sun** [2] **Wuyang Chen** [1]

Abstract

We present Lean Finder, a semantic search engine for Lean and mathlib that understands mathematicians' intents. Due to challenges of locating relevant theorems and the steep learning curve of Lean 4 language, the progress of formal theorem proving is slow by tedious human efforts. Recent Lean search engines, though helpful, only passively consider informalization of statements, largely overlooking the discrepancy from user queries in the real world. In contrast, we propose a user-centered semantic search tailored to the needs of working mathematicians. The key idea is to first analyze and cluster the semantics of public discussions on Lean, then fine-tune text embeddings on synthesized queries that simulate user intents. Our Lean Finder is thus encoded with a rich awareness of mathematicians' intents from different perspectives. Evaluations on both real-world queries by mathematicians and informalized statements demonstrate that our Lean Finder outperforms previous search engines by at least 19%. We promise to release both the code, model checkpoints, datasets, and the web service for our Lean Finder upon acceptance.

1. Introduction

Advances in Lean and mathlib (De Moura et al., 2015; Moura & Ullrich, 2021) are turning mathematical discovery into a collaborative and verifiable research workflow. On this foundation, provers powered by large language models (LLMs) have sprinted ahead (AlphaProof & teams, 2024; Xin et al., 2024a;b; Ren et al., 2025; Lin et al., 2025). Parallel progress in autoformalization has shifted from hand-written grammars to few-shot LLM translation, and large corpora by back-translating Lean code ("informalization") have bootstrapped even stronger translators (Wu et al., 2022; Jiang et al., 2024; Lu et al., 2024).

[1]School of Computing Science, Simon Fraser University [2]Department of Mathematics, Simon Fraser University [3]Department of Physics, Simon Fraser University. Correspondence to: Wuyang Chen <wuyang@sfu.ca>.

The second AI for MATH Workshop at the 42nd International Conference on Machine Learning, Vancouver, Canada.

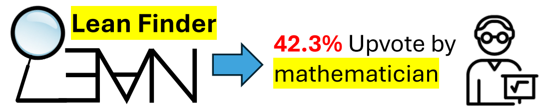

Figure 1: In the evaluation of real user queries, our Lean Finder is preferred in 42.3% of cases, compared to only 23.1% for LeanSearch (Gao et al., 2024a).

Despite these advances, state-of-the-art LLMs still cannot solve math research problems. This underscores the continued need for substantial **human efforts**, which unfortunately remain slow and labor-intensive for **two bottlenecks**. First, locating the right lemma or theorem is frustrating: search tools are rudimentary, naming conventions are often inconsistent, and high-quality Lean examples remain scarce. Second, Lean's syntax, grammar, and tactics incur a steep learning curve. Even veteran mathematicians and experienced programmers regularly report difficulties when writing Lean (Zulip, 2021b; 2020b; 2021a; 2020a).

To facilitate mathematicians, **search is vital to Lean formalization**. Recent engines take informal statements or live proof states and retrieve mathlib4 lemmas or tactic suggestions, thereby aiding human Lean users (Gao et al., 2024a;b; Tao et al., 2025; Shen et al., 2025; Ju & Dong, 2025; Asher, 2025). However, these search engines target machine translation rather than human use. They "informalize" statements from a supposedly neutral viewpoint, whereas **real users bring inherent bias**, typically seeking explanations from their own specific perspective.

Consider the two queries below. The first is an informalization of a formal statement that current Lean search engines handle (Gao et al., 2024a;b; Ju & Dong, 2025; Asher, 2025):

Query 1: Let L/K be a field extension and let x, y in L be algebraic elements over K with the same minimal polynomial. Then the K-algebra isomorphism $algEquiv$ between the simple field extensions $K(x)$ and $K(y)$ maps the generator x of $K(x)$ to the generator y of $K(y)$, i.e., $algEquiv(x) = y$.

Target Statement 1:

```
theorem algEquiv_apply {x y : L} (hx :
    IsAlgebraic K x) (h_mp : minpoly K
    x = minpoly K y) : algEquiv hx h_mp
```

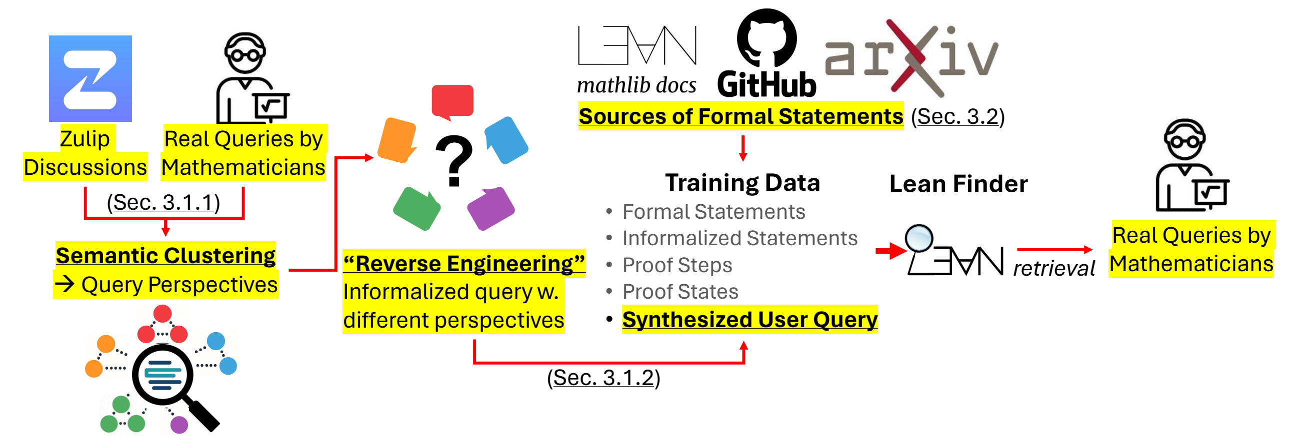

Figure 2: Overview of building our Lean Finder. We first analyze and cluster semantics of public queries by mathematicians (Section 3.1.1), and then fine-tune LLM embeddings via a large amount of synthesized queries that simulate user intents (Section 3.1.2). Moreover, our Lean Finder is further made more research-centered by including Lean 4 code from research papers and repositories (Section 3.2).

```
    (AdjoinSimple.gen K x) =
    AdjoinSimple.gen K y := by
  have hy : IsAlgebraic K y := ⟨minpoly
    K x, ne_zero hx.isIntegral, (h_mp ▷
    aeval _ _)⟩
  rw [algEquiv, trans_apply, ←
    adjoinRootEquivAdjoin_apply_root K
    hx.isIntegral, symm_apply_apply,
    trans_apply,
    AdjoinRoot.algEquivOfEq_apply_root,
    adjoinRootEquivAdjoin_apply_root K
    hy.isIntegral]
```

However, a human query in practice may look like:

Query 2: `I'm working with algebraic elements over a field extension and I have two elements, say` x `and` y `in` L. `I know` x `is algebraic over` K`, and I've managed to show that y is a root of the minimal polynomial of` x. `Does this imply that the minimal polynomials of` x `and y are actually equal? Or do I need additional assumptions to conclude that?`

Target Statement 2:

```
theorem eq_of_root {x y : L} (hx :
    IsAlgebraic K x) (h_ev :
    Polynomial.aeval y (minpoly K x) =
    0) : minpoly K y = minpoly K x :=
  ((eq_iff_aeval_minpoly_eq_zero
    hx.isIntegral).mpr h_ev).symm
```

Although both queries involve a similar mathematical context (algebraic elements x, y in a field extension L/K and their minimal polynomials), **they concern different purposes:** the first asserts the explicit behavior of an isomorphism between field extensions generated by two algebraic elements with the same minimal polynomial, whereas the second seeks to decide whether the two minimal polynomials are equal from the fact that y is a root of x's minimal polynomial. This user *latent* (motivation, perspective, level of abstraction) cannot be inferred or encoded by a purely syntactic informalization. Addressing this challenge calls for Lean search engines that can understand a mathematician's intent and discourse style, not merely surface-level matches. We defer a more rigorous analysis in Section 2.2.

Therefore, we ask our core question:

> *How to make Lean search engine user-centered and tailored to understand the intents and needs of working mathematicians?*

In this paper, we aim to build a user-centered search engine for Lean and mathlib4 that can understand the diverse intents of mathematicians. The key method is to first **analyze and cluster the semantics of public queries by mathematicians**, and then fine-tune LLM embeddings via a large amount of **synthesized queries that simulate user intents** (Section 3.1, Section 3.5). This fine-tuning strategy can enrich the awareness of real user intent from different perspectives. Moreover, our Lean Finder is further made more research-centered by including Lean 4 code from research papers and repositories (Section 3.2). We test our Lean Finder via a vast amount of retrieval evaluations with both informal statements and *real* user queries. We will release our Lean Finder retrieval system as a web service to support mathematicians coding in Lean 4. A demo is shown in Figure 4. We summarize our contributions below:

1. When evaluating on real-world queries by human mathematicians, our Lean Finder is upvoted 42.3% compared to 23.1% from LeanSearch in an LMArena-style test (Chiang et al., 2024).

2. Queried by informalized statements, Lean Finder also outperforms LeanSearch (Gao et al., 2024a) by 27.3% relative improvement on Recall@1.

3. We release the largest dataset for informal-formal pairs, with 1222600 synthesized user query, 244521 informalized statements, and 254811 Lean code snippets.

2. Motivation

2.1. Background: Search Engine for Lean and Mathlib

Lean (De Moura et al., 2015; Moura & Ullrich, 2021) is an interactive theorem prover based on dependent type theory that lets mathematicians write proofs in a rigorously checkable language. Its community-run library mathlib4 now contains over 210k theorems and 100k definitions, orders of magnitude more formal "entry points" than most programming platforms (Python exposes only 89 built-in functions; while the C++ standard library has 105 headers). **Finding the right item inside the ocean of mathlib4 is hard**: the official #find, library_search, and Loogle tools rely on exact names or goal states and quickly miss relevant results when naming conventions drift or examples are sparse. Recent LLM-powered search engines (Gao et al., 2024a;b; Yang et al., 2023; Tao et al., 2025; Shen et al., 2025; Ju & Dong, 2025; Asher, 2025; Zhu et al., 2025) embed informal queries or proof goals and retrieve matching Lean statements, but they are optimized for embedding similarity, not for the rich, shifting intents of working mathematicians. In short, sheer library scale and rudimentary search together make locating lemmas a first-order pain point, even before one tackles Lean's non-trivial syntax and tactic language.

2.2. Current Search Engines Are Misaligned with the Practical Needs of Mathematicians

Despite notable progress, current Lean search engines are optimized for informalized statements, instead of questions mathematicians actually pose. Most systems embed informal descriptions (natural languages) of statements or proof states, together with possible context or dependencies, then retrieve matching lemmas or tactics (Yang et al., 2023; Gao et al., 2024a;b; Tao et al., 2025; Shen et al., 2025). Because these informalizations are generated by LLMs rather than user logs, we have no guarantee that their wording, granularity, and purpose align with the queries real Lean practitioners type. **This unexamined domain gap between LLM-generated paraphrases and authentic human queries has so far been largely ignored**, leaving retrieval systems potentially well-tuned to wrong distributions.

To justify this domain gap, we visualize distributions of different queries embeddings generated via all-MiniLM-L6-v2 (Wang et al., 2020) sentence-transformers and PCA. As shown in Figure 3, the cluster of LLM-generated informalizations (blue) only spans over a subspace of the cluster by queries collected from Zulip and Github (green), highlighting a clear domain gap between real user queries and surrogate ones on which current Lean search engines are trained. By contrast, embeddings of our synthesized user queries (Section 3.1) are much more aligned with the human cluster, suggesting that they better capture the practical needs of Lean mathematicians.

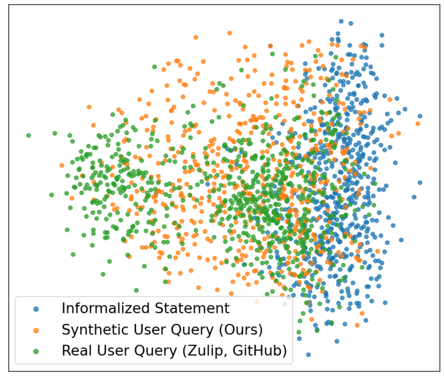

Figure 3: PCA of different queries. User queries are from Zulip and Github; Lean Finder considers our synthesized user query (Section 3.1), and informalization is from Herald (Gao et al., 2024b)

.

3. Methods

3.1. User-centered Query Synthesis

Section 2.2 underscores that effective Lean retrieval for mathematicians requires authentic user queries, yet assembling such data at scale is obstructed by two bottlenecks:

1. The volume of publicly available Lean questions by real users is tiny (for example, we only fetched 693 answerable user queries from Zulip/GitHub), which is orders of magnitude smaller than the billions-token corpora consumed by modern LLMs;

2. Even when such real user queries exist, tracing the precise answer (formal statement) that eventually resolves each query is often infeasible, due to open math problems and evolving Lean/mathlib4 development.

Therefore, preparing a large fully annotated set of genuine user queries is unrealistic. Instead, we propose a novel reverse strategy to synthesize a large amount of diverse and realistic queries. The **core idea** is: based on different perspectives that mathematicians ask Lean/mathlib4-related questions (Section 3.1.1), we prompt LLM generation to simulate mathematicians' intents (Section 3.1.2).

3.1.1. Semantic Analysis of Real Human Query

The first step for user query synthesis is to systematically collect real user discussions, and then analyze the diverse semantics and intents of mathematicians coding in Lean 4.

1. Collect User Discussions. To achieve this semantic analysis, we utilize Lean Zulip Chat[1], the primary discussion forum for Lean 4, as well as GitHub, as the main sources of real-world queries. Using the Zulip API, we extract user discussions, prioritizing high-quality and relevant content by targeting five highly active and thematically relevant channels: *new members*, *lean4*, *mathlib4*, *Is there code for X*, and *metaprogramming/tactics*.

For each thread within these channels, we retain up to the first five user messages (sorted chronologically) as they generally center around the context and core idea, and subsequent replies are more likely off the topic.

Subsequently, we prompt GPT-4o to filter these messages, retaining only discussions answerable by a formal Lean 4 statement, and paraphrasing the main query articulated within the initial messages. This procedure yields a refined collection of 693 high-quality discussions (600 from Zulip[2], 93 from GitHub).

2. Cluster User Intents. Mathematicians may ask Lean/mathlib4-related questions from different perspectives (see examples in Section 1 and our analysis in Figure 3), therefore we need to identify the distinct types of queries collected from Zulip/Github. We first prompt OpenAI o3 to bootstrap the initial set of clusters from a subset of collected queries. Then, the same model is fed with more user discussions and is prompted to iteratively update existing clusters or introduce new clusters if necessary. See Appendix E for prompts we used. This process results in five distinct and semantically meaningful clusters of queries, as shown in Table 1, capturing a comprehensive range of intents from which mathematicians ask about Lean/mathlib4.

3.1.2. Query Synthesis with User Intents

Based on semantic clusters of real queries in Table 1, we synthesize a large-scale dataset of queries simulating diverse user intents. Although it is typically challenging to resolve a query with precise formal statements in Lean, we can still **reversely synthesize the query, instead of the ground truth**. Specifically, we assume each formal Lean/mathlib4 statement could be the ground-truth answer to some unknown user queries, and now the key is to simulate or synthesize queries of realistic user intents.

We aim to instruct the LLM to generate plausible user queries that align with the specified cluster's perspective while remaining grounded in the given formal content. Concretely, we prompt GPT-4.1 mini with: (i) the Lean 4 formal statement, (ii) its informalization (Section 3.4), (iii) cluster information (cluster name, semantic definition, and representative 10-shot examples). See Figure 9 for our prompt. For each formal statement, we iterate over all five clusters in Table 1. This procedure yields a total of 1,222,600 synthetic queries distributed across five distinct intent clusters, enabling fine-tuning of embedding models better to meet mathematicians' needs on Lean search.

[1] https://leanprover.zulipchat.com/

[2] Due to privacy concerns, we do not release Zulip collections.

3.2. Research-driven Data Collection

While mathlib4 remains the primary source of formal statements for most Lean 4 search engines, it does not capture the full breadth of formal mathematics. Many recent mathematical developments are formalized beyond mathlib4, often in repositories associated with research papers or domain-specific libraries. These statements can be more relevant to working mathematicians, especially when formalizing new results or building upon cutting-edge works.

To build a more comprehensive and representative corpus of Lean 4 statements, we consider the following sources in addition to mathlib4:

- **Research-linked repository:** GitHub projects that are part of math papers, which often contain formalizations of novel theorems and lemmas.
- **Domain-specific library:** Projects such as SciLean that cover applied mathematical domains.
- **Transitive dependency:** Any Lean 4 project dependencies required by the above repositories.

We use LeanDojo (Yang et al., 2023) to extract formal statements from collected projects. The complete list of repositories in our data is shown in Table 10 in the Appendix.

This expanded collection ensures that our search engine is equipped with the most relevant, diverse, and latest formal statements to support both Lean practice and math research.

3.3. Informalization

To support retrievals by both informalized statements and the generation of synthetic user queries (Section 3.1.2), we convert formal Lean 4 statements into natural language descriptions. Inspired by Herald (Gao et al., 2024b), for each Lean 4 statement in our dataset, we provide rich context information for GPT-4o to informalize. The context consists of the following six components:

- **Formal name:** The full name of the statement to be informalized.
- **Formal statement code:** The complete Lean 4 code, including the statement header and body.
- **Docstring:** Any human-written docstring associated with the statement in the codebase.

Table 1: Five clusters of mathematicians' intents categorized from our collections of real user queries from Zulip and GitHub.

Intent Cluster	Semantic Definition	Examples
Searching for existing code / lemmas	Whether a definition, data-structure implementation, lemma, or theorem is already available in Mathlib/Lean, or if there is an easy way to get the desired statement.	"Is there code for ...", "Do we have ...", "Is there a lemma which ...", "Where can I find ..."
Meta- / tactic-programming questions	Questions about Lean 4 metaprogramming: writing tactics or elaborators, creating macros, manipulating Expr or environments, controlling metavariables, interacting with 'simp'/'aesop'/'omega' from meta code.	"Why doesn't the code give an error about a cycle existing when using Lean.MVarId.assign?", "Is there a way to make my first approach work to get Exp out of Q(Exp) using expr% macro?"
Type-class, instance, axiom	Proofs fail, or definitions cannot be written because Lean cannot find (or should not use) certain type-class instances, or users need advice on constructing/avoiding such instances and on the correct logical meaning of such instances.	"How to define or derive instances...", "Why certain instance-search problems happen...", "Whether particular instances should exist at all..."
Proof engineering & everyday Lean usage	Concrete goals or failing scripts around practical, day-to-day proof writing in Lean.	How to: finish or shorten proofs, make simp, rw, omega, linarith, etc. succeed, rewrite or unfold expressions, handle coercions and subtypes, manipulate logical connectives ($\forall, \exists, \wedge, \vee$), pattern-match over inductives, or split cases.
Library design & large-scale formalization	Conversations that concern the construction of large-scale statements: proposing new mathematical structures or tactics, refactoring existing ones, performance trade-offs, big formalizations or theorems.	"How can I apply the coYoneda lemma or density theorem to simplicial sets valued in an arbitrary universe without restricting the variable?"

- **Neighbor statement:** The closest statement in the same file, based on positional proximity. We include its full code, as nearby statements often share related concepts. This helps the model understand the local context and inter-connectiness of the statement.

- **Dependent statements:** All the dependent statements used in the body of the statement to be informalized, along with their informalizations. This provides the necessary prerequisite knowledge and semantic dependencies for achieving high-quality informalization. See Appendix A.2 for details about acquiring dependent statements.

- **Related statement in Herald** (Gao et al., 2024b): A formal statement and its corresponding informalization from the Herald dataset that closely aligns with the target statement. This example serves as an in-context demonstration for GPT-4o, helping it generate a more accurate and stylistically consistent informalization. See Appendix A.2 for details on retrieving related statements.

3.4. Other Input Modalities

For the Lean code snippet input modality, we segment statement body using newline characters (`\n`). This slicing strategy helps preserve the semantic structure of the original proof. The input sequences in our dataset contain 2 to 4 code segments, omitting single-line inputs, as individual lines are often trivial (e.g., `rfl`) and provide limited semantic value on their own. The statement definition input modality excludes the body of the statement, and retains only the statement header information.

We summarize our dataset by input modalities in Table 2 and by data sources in Table 3.

Table 2: Overview of our dataset by input modalities.

Input Modality	Samples	Percent
Synthesized User Query	1222600	63%
Informalized Statements	244521	12%
Lean Code Snippet	254811	13%
Formal Statements	244521	12%

Table 3: Overview of our dataset by data sources.

Source	Samples	Percent
Mathlib	238021	97%
Research-linked repository	2198	1%
Lean 4 related libraries	4302	2%

3.5. Training Lean Finder with Code Search

Our code search model, named **Lean Finder**, adopts DeepSeek-Prover-V1.5-RL 7B (Xin et al., 2024b) as the base model for fine-tuning, for its extensive pretraining on Lean 4 syntax and theorem proving tasks. As a decoder-only architecture, only the final token in each sequence has access to the full context due to the causal self-attention. Therefore, we extract the final hidden state of the last token in the last decoder layer to embed the entire sequence. The model is fine-tuned with contrastive loss on "informal query q_i – formal code c_j" pairs, which aims to align the embeddings of matching pairs in a shared embedding space.

For each training pair (q_i, c_j), we create an augmented version $q_i^* = \text{Augment}(q_i), \quad c_i^* = \text{Augment}(c_i)$. The augmentation randomly selects 15% of tokens in the input sequence, with (1) 80% of the selected tokens replaced with a special [MASK] token, (2) 10% replaced with random vocabulary tokens, (3) 10% unchanged. This augmentation encourages robust embeddings to noisy or partial inputs.

Inspired by prior works in contrastive learning and code search (Wang et al., 2023; Li et al., 2022; 2021), we use a momentum decoder and maintain four queues of negative examples: (1) unmasked queries, (2) masked queries, (3) unmasked code, (4) masked code.

Let D denote the main decoder that is updated via gradient, and D' denote the momentum decoder updated via exponential moving average of the main decoder D. For any input sequence I, we denote its normalized embedding from the main decoder as $D(I)$, and from the momentum decoder as $D'(I)$. We define four similarities:

$$\mathrm{Sim}(c_i, q_j) = D(c_i)^\top D'(q_j) \tag{1}$$
$$\mathrm{Sim}(c_i, q_j^*) = D(c_i)^\top D'(q_j^*) \tag{2}$$
$$\mathrm{Sim}(q_i, c_j) = D(q_i)^\top D'(c_j) \tag{3}$$
$$\mathrm{Sim}(q_i, c_j^*) = D(q_i)^\top D'(c_j^*) \tag{4}$$

For each informal query and formal Lean code within a batch of samples, we compute the temperature-scaled softmax similarity distributions:

$$P^{c\to q}(c_i, q_j) = \frac{\exp\left(\mathrm{Sim}(c_i, q_j)/\tau\right)}{\sum_{m=1}^{M}\exp\left(\mathrm{Sim}(c_i, q_m)/\tau\right)} \tag{5}$$
$$P^{c\to q^*}(c_i, q_j) = \frac{\exp\left(\mathrm{Sim}(c_i, q_j^*)/\tau\right)}{\sum_{m=1}^{M}\exp\left(\mathrm{Sim}(c_i, q_m^*)/\tau\right)} \tag{6}$$
$$P^{q\to c}(q_i, c_j) = \frac{\exp\left(\mathrm{Sim}(q_i, c_j)/\tau\right)}{\sum_{m=1}^{M}\exp\left(\mathrm{Sim}(q_i, c_m)/\tau\right)} \tag{7}$$
$$P^{q\to c^*}(q_i, c_j) = \frac{\exp\left(\mathrm{Sim}(q_i, c_m^*)/\tau\right)}{\sum_{m=1}^{M}\exp\left(\mathrm{Sim}(q_i, c_m^*)/\tau\right)} \tag{8}$$

where $M = B + K$, where B is the batch size and K is the queue size. τ is the temperature hyperparameter.

To mitigate semantic ambiguity in the input and reduce the impact of noisy supervision (e.g. weakly correlated query-code pairs), we employ a soft labeling strategy. Instead of relying solely on hard one-hot labels, we generate soft targets as a combination of a one-hot similarity vector and pseudo-targets derived from the momentum decoder D'. Specifically, we define the similarity between a query-code pair using the momentum decoder as:

$$\mathrm{Sim}'(q_i, c_j) = D'(q_i)^\top D'(c_j) \tag{9}$$

The soft target vector $Y_{soft}^{q\to c}$ is computed as:

$$Y_{soft}^{q\to c} = (1-\beta) * Y_{onehot}^{q\to c} + \beta * S^{q\to c} \tag{10}$$

where β is a hyperparameter, $Y_{onehot}^{q\to c}$ is a one-hot vector with a value of 1 at the index of the positive target and 0 elsewhere, and $S^{q\to c}$ is a pseudo-target distribution over all Lean 4 statement code embeddings in the queue computed via softmax over momentum-based similarities:

$$S_{ij}^{q\to c} = \frac{\exp\left(\mathrm{Sim}'(q_i, c_j)/\tau\right)}{\sum_{m=1}^{M}\exp\left(\mathrm{Sim}'(q_i, c_m)/\tau\right)}. \tag{11}$$

Let H be the cross-entropy loss, and $\mathcal{D}$ be the training set of query-code pairs, our overall loss function is defined as:

$$\mathcal{L} = \frac{1}{4}\mathbb{E}_{(c,q)\sim\mathcal{D}}\Big[\mathrm{H}(\mathbf{Y_{soft}}^{c\to q}, \mathbf{p}^{c\to q}) + \mathrm{H}(\mathbf{Y_{soft}}^{c\to q^*}, \mathbf{p}^{c\to q^*}) + \mathrm{H}(\mathbf{Y_{soft}}^{q\to c}, \mathbf{p}^{q\to c}) + \mathrm{H}(\mathbf{Y_{soft}}^{q\to c^*}, \mathbf{p}^{q\to c^*})\Big].$$

4. Experiments

4.1. LLM Settings

We utilize the OpenAI API to synthesize user queries from formal Lean statements as part of our dataset generation pipeline. Please see Appendix B for details on API configurations. For retrieval, we fine-tune DeepSeek-Prover-V1.5-RL 7B using LoRA and a query-code contrastive learning objective. Training settings are provided in Appendix C.

4.2. Evaluation Settings and Data

To evaluate the performance of our Lean Finder, we compare it with LeanSearch (Gao et al., 2024a;b), a widely used search engine for Lean 4. To the best of our knowledge, LeanSearch is currently the only search engine that supports Lean retreival for Mathlib statements using either natural language queries or Lean 4 code.

We assess retrieval performance using the following metrics, both higher the better:

- **Recall@K**: This measures the proportion of relevant results within the top K retrieved results.
- **Mean Reciprocal Rank (MRR)**: This calculates the average of the reciprocal ranks of the first relevant result for each query. It is defined as: $\mathrm{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\mathrm{rank}_i}$, where N is the number of queries, and rank_i is the rank position of the first relevant result for query i.

We evaluate our Lean Finder across four input modalities: *informalized statement*, *synthetic user query*, *statement definition*, and *Lean code snippet*.

Informalized Statement. To ensure a fair comparison, we impose two constraints: 1) The ground-truth formal statement *must* exist in the LeanSearch database (otherwise LeanSearch would always fail to retrieve it). 2) The corresponding informal query *must not* appear in either the LeanSearch database or our training data (otherwise LeanSearch would retrieve it perfectly, and our model would already have seen a near-duplicate). Therefore, we randomly select 150 Lean 4 statements from LeanSearch's database, and informalize them using a simplified prompt without additional context (dependent theorems, related theorems, etc.) used in our model. These newly informalized statements constitute a fair test set for comparison.

Synthetic User Query. For the synthetic user query modality, we generate 150 queries evenly distributed across five clusters, using the 30 of the newly informalized statements as part of the prompt for LeanSearch comparison.

Statement Definition. To evaluate with the statement definition, we sample 100 Lean 4 statements from our dataset and their corresponding definitions that are not in our training set of Lean Finder. To simulate noisy queries and test model robustness, we randomly replace 20% of the words in the query with tokens sampled from the vocabulary. To avoid invalid characters introduced by the retrieval model's tokenizer during augmentation, we use a custom tokenizer when augmenting the statement definitions.

Lean Code Snippet. Finally, for the code snippet modality, we sample 100 examples of code snippets from our dataset not present in our training set, and we do not apply augmentation.

All evaluation on LeanSearch is done manually using their web service[3].

4.3. Results on Synthetic Queries

Table 4: Retrieval by informalized statement.

Models	R@1	R@5	R@10	MRR
LeanSearch	0.487	0.700	0.773	0.599
Lean Finder (ours)	0.620	0.893	0.933	0.735

Table 5: Retrieval by synthetic user query.

Models	R@1	R@5	R@10	MRR
LeanSearch	0.333	0.593	0.713	0.465
Lean Finder (ours)	0.407	0.693	0.780	0.537

Table 6: Retrieval by statement definition.

Models	R@1	R@5	R@10	MRR
LeanSearch	0.620	0.840	0.870	0.716
Lean Finder (ours)	0.650	0.940	0.960	0.784

Table 7: Retrieval by Lean code snippets

Models	R@1	R@5	R@10	MRR
LeanSearch	0.090	0.260	0.330	0.182
Lean Finder (ours)	0.56	0.81	0.86	0.678

[3]https://leansearch.net/

From results in Table 4, 5, 6, and 7, we observe that our Lean Finder consistently outperforms LeanSearch across all evaluated input modalities. Notably, for informalized statements, synthetic user queries, and Lean code snippets, our model achieves substantial performance gains. These improvements highlight the model's superior ability to capture user intent and generalize across diverse query formats, making it a more versatile and effective tool for supporting Lean 4 developers in real-world theorem proving workflows.

4.4. Results on Real User Queries

Table 8: Retrieval by real user query. "Preferred Method": by which method the user-preferred answer is retrieved. "Tied": retrievals by LeanSearch (Gao et al., 2024a) and ours are comparable.

Preferred Method	Count	Percentage
LeanSearch	12	23.1%
Lean Finder (ours)	22	42.3%
Tied	18	34.6%

Table 9: User preference (Table 8) by intents. Cluster 1: Searching for existing code/lemmas. Cluster 3: Type-class, instance, axiom. Cluster 4: Proof engineering & everyday Lean usage.

Clusters	Ours Preferred	LeanSearch Preferred	Tied
Cluster 1	11	3	3
Cluster 3	1	1	0
Cluster 4	10	8	15

To further evaluate Lean Finder's effectiveness in real-world scenarios, we conduct a user study using Lean queries made by humans. We select 52 high-quality answerable user queries from GitHub and retrieve the top-3 statements from both LeanSearch and our Lean Finder for each query. The evaluation follows a style similar to LMArena (Chiang et al., 2024), where the order of the retrieval method is randomly shuffled for each query. As shown in Table 8, our Lean Finder was preferred in 42.3% of cases, substantially outperforming LeanSearch (23.1%), demonstrating the model's superior ability to capture user intent and provide satisfying answers to real user queries.

We further analyze this user preference by clustering these evaluated real user queries, and show user preference per cluster in Table 9. We find that our Lean Finder demonstrates stronger performance in Clusters 1 and 4.

4.5. Lean Finder's User Interface

To facilitate Lean users, we also develop and deploy our user interface, as shown in Figure 4. We retrieve formal statements and also report their matching scores. We turn our web service offline to avoid disclosing identity information.

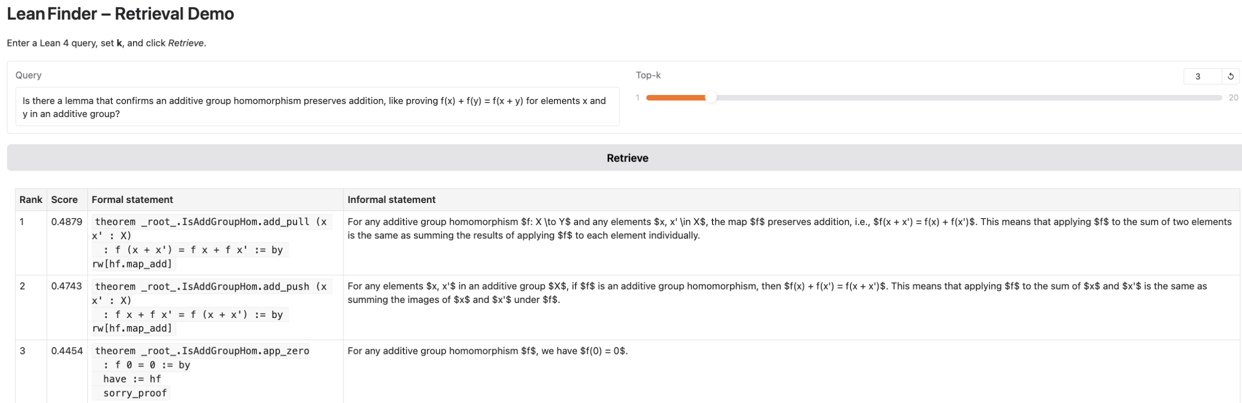

Figure 4: Web user interface of our Lean Finder.

5. Related Works

Code Search and Lean Retrieval. Code search, aimed at retrieving the most semantically relevant code snippets from a codebase according to a specified natural language query, is a common activity that plays an important role in software development (Husain et al., 2019; Wang et al., 2023) and aligns closely with premise selection: both problems require retrieving a small set of code blocks that are helpful for downstream tasks. Early methods relied on lexical heuristics (Alama et al., 2014; Urban, 2004), while recent work leverages neural encoders (Irving et al., 2016; Wang & Deng, 2020; Mikuła et al., 2023; Yeh et al., 2023) and dense retrieval models (Karpukhin et al., 2020; Qu et al., 2020) by learning semantic embeddings. In formal domains like Lean, retrieval systems like LeanSearch (Gao et al., 2024a;b) assist users by allowing them to search for relevant mathlib statements using natural language queries. These systems primarily function by matching the natural language translation of a formal statement to other formal statements in mathlib. While effective in some settings, this approach supports only narrow input types and treats informal queries as approximate paraphrases of formal code and focuses on surface-level alignment, without capturing the diverse intents behind real-world user questions or the broader context in which they arise. Lean Finder addresses this gap by introducing a user-centered, intent-aware retrieval framework for Lean that supports a broad range of input modalities and explicitly bridges the gap between the natural queries posed by mathematicians and the formal statements stored in Lean projects. By modeling real user intent and query diversity, Lean Finder achieves consistently strong retrieval performance across all modalities and demonstrates substantial gains over existing systems in a user study based on real Lean queries.

AutoFormalization. Autoformalization, the automated translation of informal mathematics into formal proof-assistant code, has become a key strategy for addressing the scarcity of high-quality formal data. Early neural encoders for statement formalization (Wang et al., 2018) have been eclipsed by LLM-based pipelines that translate at scale with few-shot prompting or fine-tuning (Wu et al., 2022; 2024; Xin et al., 2024a;b). Systems such as DeepSeek-Prover and InternLM2.5-StepProver formalize vast Lean corpora for expert iteration (Xin et al., 2024a; Wu et al., 2024), while others generate synthetic formal proofs without informal seeds (Wu et al., 2020; Wang & Deng, 2020; Poesia et al., 2024). Projects such as MMA, Lean-STaR, TheoremLlama, and Lean Workbook create natural-language/formal-language (NL–FL) pairs to mitigate data scarcity (Jiang et al., 2023; Lin et al., 2024; Wang et al., 2024; Ying et al., 2024); yet LLM brittleness still introduces subtle logical errors. Recent LLM tools embedded in user workflows (e.g., LeanDojo, LeanCopilot, LLM-Step) further streamline manual formalization by suggesting premises and proof tactics (Yang et al., 2023; Song et al., 2023; Welleck & Saha, 2023). Our work tackles the complementary challenge of autoformalization: synthesizing intent-rich, user-style queries from formal Lean statements to better serve practitioners. Unlike previous informalization approaches that simply translate formal statements into natural language, our pipeline explicitly models user intent and query phrasing. Injecting this intent-aware signal during fine-tuning yields significant retrieval improvements on real user queries, and bridges the gap between informal user needs and the formal knowledge encoded in Lean libraries.

6. Conclusion

We present Lean Finder, a significant advancement in the development of search tools for formal mathematics. By centering on real user intent and leveraging synthesized queries, semantic clustering, and fine-tuned embeddings, it bridges the gap between mathematicians' informal reasoning and formal theorem proving in Lean. Its superior performance over existing search tool demonstrate both practical utility and academic impact. Lean Finder not only enhances accessibility to mathlib4 but also lays the groundwork for future intelligent systems that assist in rigorous, collaborative mathematical discovery.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Alama, J., Heskes, T., Kühlwein, D., Tsivtsivadze, E., and Urban, J. Premise selection for mathematics by corpus analysis and kernel methods. *Journal of Automated Reasoning*, 52:191–213, 2014.

AlphaProof and teams, A. AI achieves silver-medal standard solving international mathematical olympiad problems. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/, 2024.

Asher, J. Leanexplore: A search engine for lean 4 declarations. https://github.com/justincasher/lean-explore/blob/main/LeanExplore.pdf, 2025.

Chiang, W.-L., Zheng, L., Sheng, Y., Angelopoulos, A. N., Li, T., Li, D., Zhu, B., Zhang, H., Jordan, M., Gonzalez, J. E., et al. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*, 2024.

De Moura, L., Kong, S., Avigad, J., Van Doorn, F., and von Raumer, J. The lean theorem prover (system description). In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pp. 378–388. Springer, 2015.

Gao, G., Ju, H., Jiang, J., Qin, Z., and Dong, B. A semantic search engine for mathlib4. *arXiv preprint arXiv:2403.13310*, 2024a.

Gao, G., Wang, Y., Jiang, J., Gao, Q., Qin, Z., Xu, T., and Dong, B. Herald: A natural language annotated lean 4 dataset. *arXiv preprint arXiv:2410.10878*, 2024b.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

Irving, G., Szegedy, C., Alemi, A. A., Eén, N., Chollet, F., and Urban, J. DeepMath—deep sequence models for premise selection. 2016.

Jiang, A. Q., Li, W., and Jamnik, M. Multilingual mathematical autoformalization. *arXiv preprint arXiv:2311.03755*, 2023.

Jiang, A. Q., Li, W., and Jamnik, M. Multi-language diversity benefits autoformalization, 2024.

Ju, H. and Dong, B. Mirb: Mathematical information retrieval benchmark. *arXiv preprint arXiv:2505.15585*, 2025.

Karpukhin, V., Oguz, B., Min, S., Lewis, P. S., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pp. 6769–6781, 2020.

Li, C., Wang, Z., Bai, Y., Duan, Y., Gao, Y., Hao, P., and Wen, Z. Formalization of algorithms for optimization with block structures, 2025a. URL https://arxiv.org/abs/2503.18806.

Li, C., Xu, S., Sun, C., Zhou, L., and Wen, Z. Formalization of optimality conditions for smooth constrained optimization problems, 2025b. URL https://arxiv.org/abs/2503.18821.

Li, J., Selvaraju, R. R., Gotmare, A. D., Joty, S., Xiong, C., and Hoi, S. Align before fuse: Vision and language representation learning with momentum distillation, 2021. URL https://arxiv.org/abs/2107.07651.

Li, J., Li, D., Xiong, C., and Hoi, S. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation, 2022. URL https://arxiv.org/abs/2201.12086.

Lin, H., Sun, Z., Yang, Y., and Welleck, S. Lean-star: Learning to interleave thinking and proving, 2024. URL https://arxiv.org/abs/2407.10040.

Lin, Y., Tang, S., Lyu, B., Wu, J., Lin, H., Yang, K., Li, J., Xia, M., Chen, D., Arora, S., et al. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025.

Lu, J., Wan, Y., Liu, Z., Huang, Y., Xiong, J., Liu, C., Shen, J., Jin, H., Zhang, J., Wang, H., Yang, Z., Tang, J., and Guo, Z. Process-driven autoformalization in Lean 4. *arXiv preprint arXiv:2406.01940*, 2024.

Manthe, S. A formalization of borel determinacy in lean, 2025. URL https://arxiv.org/abs/2502.03432.

Mikuła, M., Antoniak, S., Tworkowski, S., Jiang, A. Q., Zhou, J. P., Szegedy, C., Kuciński, Ł., Miłoś, P., and Wu, Y. Magnushammer: A transformer-based approach to premise selection. *arXiv preprint arXiv:2303.04488*, 2023.

Monnerjahn, L. Formalisation of constructable numbers. 2024.

Monticone, P. Formalising fermat's last theorem for exponent 3 in lean. 2024.

Moura, L. d. and Ullrich, S. The Lean 4 theorem prover and programming language. 2021.

Poesia, G., Broman, D., Haber, N., and Goodman, N. D. Learning formal mathematics from intrinsic motivation. *arXiv preprint arXiv:2407.00695*, 2024.

Qu, Y., Ding, Y., Liu, J., Liu, K., Ren, R., Zhao, W. X., Dong, D., Wu, H., and Wang, H. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2010.08191*, 2020.

Ren, Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Shen, Z., Huang, N., Yang, F., Wang, Y., Gao, G., Xu, T., Jiang, J., He, W., Yang, P., Sun, M., et al. Real-prover: Retrieval augmented lean prover for mathematical reasoning. *arXiv preprint arXiv:2505.20613*, 2025.

Song, P., Yang, K., and Anandkumar, A. Towards large language models as copilots for theorem proving in lean. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023. URL https://openreview.net/forum?id=C9X5sXa2k1.

Tao, Y., Liu, H., Wang, S., and Xu, H. Assisting mathematical formalization with a learning-based premise retriever. *arXiv preprint arXiv:2501.13959*, 2025.

Urban, J. MPTP—motivation, implementation, first experiments. *Journal of Automated Reasoning*, 33:319–339, 2004.

Wang, M. and Deng, J. Learning to prove theorems by learning to generate theorems. 2020.

Wang, Q., Kaliszyk, C., and Urban, J. First experiments with neural translation of informal to formal mathematics. In *Conferences on Intelligent Computer Mathematics (CICM)*, 2018.

Wang, R., Zhang, J., Jia, Y., Pan, R., Diao, S., Pi, R., and Zhang, T. Theoremllama: Transforming general-purpose llms into lean4 experts. *arXiv preprint arXiv:2407.03203*, 2024.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020. URL https://arxiv.org/abs/2002.10957.

Wang, Y., Le, H., Gotmare, A. D., Bui, N. D., Li, J., and Hoi, S. C. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023.

Welleck, S. and Saha, R. llmstep: LLM proofstep suggestions in lean. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023. URL https://openreview.net/forum?id=ODOJuAM4Qj.

Wu, Y., Jiang, A. Q., Ba, J., and Grosse, R. Int: An inequality benchmark for evaluating generalization in theorem proving. *arXiv preprint arXiv:2007.02924*, 2020.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with large language models. 2022.

Wu, Z., Huang, S., Zhou, Z., Ying, H., Wang, J., Lin, D., and Chen, K. InternLM2.5-StepProver: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 2024.

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. *arXiv preprint arXiv:2405.14333*, 2024a.

Xin, H., Ren, Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., et al. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024b.

Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R. J., and Anandkumar, A. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612, 2023.

Yeh, E., Hitaj, B., Owre, S., Quemener, M., and Shankar, N. CoProver: A recommender system for proof construction. *arXiv preprint arXiv:2304.10486*, 2023.

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, 2024. URL https://arxiv.org/abs/2406.03847.

Zhu, T., Clune, J., Avigad, J., Jiang, A. Q., and Welleck, S. Premise selection for a lean hammer. *arXiv preprint arXiv:2506.07477*, 2025.

Zulip. How does a noob learn to write tactics. https://leanprover-community.github.io/archive/stream/113489-new-members/topic/How.20does.20a.20noob.20learn.20to.20write.20tactics.3F.html, 2020a.

Zulip. Naming conventions for definitions. https://leanprover-community.github.io/archive/stream/113488-general/topic/Noun.2Fadjective.2Fverb.20naming.20conventions.20for.20defs.20in.20mathlib.html, 2020b.

Zulip. Improving documentation. https://leanprover-community.github.io/archive/stream/113488-general/topic/Improving.20documentation.html, 2021a.

Zulip. Library search failures. https://leanprover-community.github.io/archive/stream/113488-general/topic/library_search.20failures.html, 2021b.

A. Details for Dataset Construction

A.1. Data Source Details

Table 10: Lean sources included in our dataset beyond mathlib4.

Project name	Category	URL
Mathlib4	Domain-specific library	https://github.com/leanprover-community/Mathlib4
SciLean	Domain-specific library	https://github.com/lecopivo/SciLean
A formalization of Borel determinacy in Lean (Manthe, 2025)	Research-linked repository	https://github.com/sven-manthe/A-formalization-of-Borel-determinacy-in-Lean
Optlib (Li et al., 2025a;b)	Research-linked repository	https://github.com/optsuite/optlib
Formalising Fermat's Last Theorem for Exponent 3 (Monticone, 2024)	Research-linked repository	https://github.com/riccardobrasca/flt3
Formalisation of constructable numbers (Monnerjahn, 2024)	Research-linked repository	https://github.com/Louis-Le-Grand/Formalisation-of-constructable-numbers
Plausible	Transitive dependency	https://github.com/leanprover-community/plausible
ProofWidgets4	Transitive dependency	https://github.com/leanprover-community/ProofWidgets4
Aesop	Transitive dependency	https://github.com/leanprover-community/aesop
Batteries	Transitive dependency	https://github.com/leanprover-community/batteries

A.2. Informalization Details

To support the inclusion of dependent statements during informalization, we construct a directed acyclic graph in which each node represents a Lean 4 statement and edges capture dependency relationships between statements. We ensure all dependencies of a given statement are informalized before the statement itself.

We use DeepSeekProver to retrieve the most relevant statements from Herald, and use the top-matching informal statement along with its corresponding formal statement as the related statement in the prompt for informalization.

B. OpenAI API Configurations

We prompt OpenAI models during data collection and generation, and we describe the detailed configurations below.

Informalization. We leverage the `GPT-4o` API to convert formal Lean 4 statements into informal statements (Section 3.4). The temperature is set to 0.2 and the maximum output token limit is 1000. Inspired by Herald (Gao et al., 2024b), the prompt used for informalization is shown in Figure 5.

User Query Filtering. We employ `GPT-4o` to filter out user queries that cannot be addressed by any Lean 4 formal statement (Section 3.1.1). The temperature is set to 0.0 with a maximum of 500 output tokens. The prompted used for query filtering is shown in Figure 6.

User Intent Clustering. To cluster user intents (Section 3.1.1), we use the `o3` API. The process involves two stages:

- **Bootstrap Categorization.** An initial set of representative user queries is clustered using a temperature of 1.0 and a token limit of 20000. The prompt used for initial cluster generation is shown in Figure 7.
- **Progressive Clustering.** Remaining queries are progressively categorized with the same temperature and token settings. The prompt used for this part is shown in Figure 8.

User Query Generation. To synthesize diverse user queries (Section 3.1.2), we use the `GPT-4.1 mini` API with a temperature of 0.7 and a maximum output of 1000 tokens. The prompt used is shown in Figure 9.

C. Training Details

We fine-tune our Lean Finder based on the DeepSeek-Prover-V1.5-RL 7B (Xin et al., 2024b), using a query–code contrastive learning objective. We apply LoRA (Hu et al., 2021) with a rank $r = 32$ and $\alpha = 64$. We train Lean Finder for 2 epochs with a per-GPU batch size of 10, gradient accumulation steps of 6, and 4 NVIDIA A100 GPUs. We use the AdamW optimizer with a learning rate of 2×10^{-5} and $\epsilon = 1 \times 10^{-8}$. The learning rate is linearly warmed up over the first 1,210 steps, then decayed to 0 using a cosine schedule. The momentum decoder is updated using a momentum coefficient of 0.99, and the temperature τ for scaling the similarity distributions is set to 0.7. We set the momentum queue size to $K = 640$. For soft labeling, the weight β begins at 0 and is linearly increased to 0.4 over the course of training. Due to imbalanced input modalities in our training data (Table 2), we adopt a weighted sampler assigning higher sampling probabilities to examples from the underrepresented modality.

D. Privacy and Data Release

To protect user privacy, we do not release any real user discussion data collected from Lean Zulip. All training of Lean Finder involving Zulip-derived queries are based solely on synthetic user queries that are reverse-engineered from formal Lean statements. We will only release these synthetic queries, along with queries from other input modalities. No identifiable information or original messages from Zulip users are included in our datasets.

E. Prompts

```
You are an expert mathematician and an expert in Lean and Mathlib.
**Instruction**:
Your task is to first translate the formal theorem provided below into an informal statement that is more accessible to mathematicians and written in LaTeX. There are six parts of information as inputs:
1. Formal name: The formal name of the theorem you need to translate.
2. Formal statement: The formal statement of the theorem you need to translate.
3. Docstrings: A list of natural language comments written by humans in the theorem
4. Neighbor statement: The statement located in the same file and closest in position to the formal theorem you need to translate.
5. Dependent theorems: These are JSON list of dependent theorems at which the theorem you need to translate uses, and their informal statements and name.
6. Related statement: This is a JSON object for the statement from mathlib4 that has related meaning as the current theorem at which you need to translate. The related statement formal statement, informal statements, and informal name are provided.

Then create an informal name. Use the provided formal name of the statement according to the naming conventions. Utilize the informal statement written in the first task. Make sure you follow the principles of informal naming when creating informal names. Principles of informal statements should also be followed as much as possible.

**Principles of Informal Statements**:
The informal statement should be written using human-used mathematical notations and formulas in LaTeX as much as possible, while also explaining the meaning of the symbols involved.
Explain more detailed mathematical setup only if the definition appearing in the statement is not commonly accepted. Both the inputs and values of the definition should be expressed using mathematical formulas as much as possible.

*Example*:
DO NOT use "Real.log";
Use $\log$ instead.

**Principles of Informal Names**:
Emphasize the core concepts in the definition. The definition name should not merely list concepts; use words that indicate logical relationships and clearly state the conclusion.

*Example*:
Use "A equals B" or "A implies B" (or simply "$A = B$" and "$A \to B$"), instead of "theorem of A and B" when the theorem states the result of A = B or A → B. Both the inputs and values of the definition should be expressed using mathematical formulas as much as possible.

 **Input**:
 *Formal name*: <>
 *Formal Statement*: <>
 *Docstring*: <>
 *Neighbor statement:* <>
 *Dependent theorems*: <>
 *Related theorem*: <>

 **Output**:
 Return *only* a JSON object with keys
  "informal name"
  "informal statement"
```

Figure 5: Prompt used for Lean statement informalization.

You are an expert at Lean 4 and an experienced user of Lean Zulip Chat.
Instruction:
Your task is to first read the excerpt provided below that contains the first 5 messages from a discussion thread in Lean Zulip Chat and then decide to accept or reject the excerpt based on the following criteria:
1. You should accept the excerpt if the main user question in the excerpt can be answered at least partially by providing a Lean 4 statement.
2. You should accept the excerpt if showing a pre-existing Lean 4 statement will move the discussion forward in a meaningful way.
3. You should reject the excerpt if none of the above is true for this excerpt.
4. When uncertain, use your best judgment.
Then identify the main question in this discussion thread based on the excerpt provided below. The main question is the one that the user who initiated the discussion wants an answer for.

Examples (not exhaustive):
ACCEPT:
• The user is asking “Is there a lemma/theorem/definition that …?
• The user needs a property that already exists as a lemma (e.g. “`map_prod` preserves multiplication”).
• The user doesn't directly ask for a statement, but providing an example statement to the user can address part of the problem.
REJECT:
• Library infrastructure, version bumps, build tools, CI, style.

Input:
{input_block}

Output:
Return *only* a JSON object with keys
  “decision”
  “main_question”
Where the value for decision must be *exactly* one word ACCEPT or REJECT

Figure 6: Prompt used user query filtering.

```
You are an expert at Lean 4 and an experienced user of Lean Zulip Chat.
   **Instruction**:
You will be provided below with 50 excerpts that contain the first few messages from discussion threads in Lean Zulip Chat and the main user question that is being
asked in that discussion thread. Each excerpt has been filtered so that the main user question can be answered at least partially by a statement written in Lean 4 or
showing a pre-existing Lean 4 statement will move the discussion forward in a meaningful way. You need to do the following task:
Group the following 50 excerpts into clusters by their query purpose/type, and follow the instruction below:
* Each cluster should represent a distinct type of query based on the underlying intent or perspective of the main question in the excerpt.
* Give each cluster a descriptive name (a few words) that summarizes that query type.
* For each cluster, give a detailed description of what the cluster represents. This should include the common properties shared by the queries that belong in these
clusters, and the description of the underlying intent or perspective of the queries made by the user. This must be detailed enough, such that a person reading this
description can create new queries for a Lean 4 statement that belong in this cluster.
* For each cluster, include at most 5 representative example queries that best define this cluster in the description. The chosen example queries must be the
main_question provided in the input for the excerpt.

The input will be a list of excerpts, where each excerpt will have the following contents:
1. channel: This contains the name of the channel that the excerpt is collected from in the Lean Zulip Chat.
2. topic: This contains the name of the discussion thread the excerpt is collected from.
3. main_question: This contains the main user question that is being asked in the excerpt.
4. messages: This contains a list of initial messages from the discussion thread. Each message in the list contains the full name of the sender and the content of the
message.

**Input**:
{input_excerpts}
**Output**:
Return only a JSON object with a key "clusters", and the value for the key is a list of cluster objects.
Each cluster object must have keys: "cluster_name", "cluster_description", and "examples".
The value of the key "cluster_name" should be the name of that cluster.
The value of the key "cluster_description" should be the detailed description of that cluster.
The value for the key "examples" should be a list of main_question from the chosen excerpts in the input.

*Example output JSON*:
{
   "clusters": [
      {
         "cluster_name": "the name of the first cluster",
         "cluster_description": "the detailed description for the first cluster",
         "examples": ["main_question 1", "main_question 2"]
       },
       {
         "cluster_name": "the name of the second cluster",
         "cluster_description": "the detailed description for the second cluster",
         "examples": ["main_question 3"]
       }
   ]
```

Figure 7: Prompt used for generating initial clusters of user queries.

You are an expert at Lean 4 and an experienced user of Lean Zulip Chat.
Instructions:
You will be provided below with 25 excerpts that contain the first few messages from discussion threads in Lean Zulip Chat and the main user question that is being asked in that discussion thread. Each excerpt has been filtered so that the main user question can be answered at least partially by a statement written in Lean 4 or showing a pre-existing Lean 4 statement will move the discussion forward in a meaningful way. You will also be provided with the current clusters of queries based on previous excerpts. Each cluster represents a distinct type of query based on the underlying intent or perspective of the main question in the excerpt. You need to do the following:
Task A -- Fitting new excerpts :
1. For each input excerpt, try to fit the excerpt into an existing cluster based on the cluster's description and representative examples provided for queries in that cluster.
2. If new excerpts fit into an existing cluster of queries and the new excerpts reveal nuances that the current description of the cluster does not capture, you should append at most one new sentence to an existing cluster description.
3. If new excerpt fit into an existing cluster of queries and the main_question in the new excerpts reveal nuances that the current examples for that cluster does not capture, you should add at most one new example to an existing cluster examples.
4. DO NOT remove any description and examples in the existing cluster. You are only allowed to add new ones when needed.
If the excerpt DOES NOT fit into an existing cluster, do *Task B*, otherwise skip *Task B*

Task B – New cluster proposal (if needed)
1. If the input excerpt doesn't fit into any existing cluster, then you should propose one new cluster.
2. The new cluster must have a descriptive name and a detailed description of what the cluster represents. The description should include the property of the query in this new cluster, and the description of underlying intent or perspective of the query made by the user. This must be detailed enough, such that a person reading this description can create new queries for a Lean 4 statement that belongs in this cluster.
3. The new cluster must include at most 5 representative example queries that best define this cluster. The chosen example queries must be the main_question provided in the input for the excerpt. Each included example must illustrate different situations within the cluster, so that a reader can generalise and invent many other queries that would still belong here.

Important:
DO NOT remove existing clusters, even if they are not used by the new excerpts.
DO NOT remove existing description and examples from existing clusters, You are only allowed add new clusters or add new description/examples to existing clusters.

The input will be a list of excerpts and a list of clusters.
Each excerpt will have the following contents:
1. channel: This contains the name of the channel that the excerpt is collected from in the Lean Zulip Chat.
2. topic: This contains the name of the discussion thread the excerpt is collected from.
3. main_question: This contains the main user question that is being asked in the excerpt.
4. messages: This contains a list of initial messages from the discussion thread. Each message in the list contains the full name of the sender and the content of the message.

Each cluster will have the following contents:
1. cluster_name: the name of the cluster
2. cluster_description: the detailed description of what this cluster represents
3. examples: representative examples in this cluster

Inputs:
List of excerpts:
{list_of_excerpts}
List of clusters:
{list_of_clusters}

Output:
Return only a JSON object with a key "clusters", and the value for the key is a list of cluster objects.
Each cluster object must have keys: "cluster_name", "cluster_description", and "examples".

Figure 8: Prompt used for progressive clustering of user queries.

You are an expert Lean 4 user and an experienced participant in the Lean Zulip chat. You know how people usually phrase questions when they are searching for existing Lean statements..
Instructions:
You will be given:
1. The name, description, and 10 canonical examples of a specific cluster of queries from Lean Zulip or GitHub. The queries in this cluster share the same underlying intent or perspective, precisely captured by the description.
2. A single Lean 4 formal statement, formal name, informal name, and the natural language description of the statement.
Write one realistic user query that:
1. Clearly belongs in this cluster –- the intent and perspective of the query should mirror the description and examples.
2. Could be answered at least partially by showing the Lean 4 statement you were given, or would be moved forward in a meaningful way by that statement.
3. Does not explicitly reveal the statement.
4. Feels like a genuine message a mathematician or Lean 4 user would post (natural tone, sensible level of detail).
5. Stands alone – no references to “the cluster” or to these instructions.

Input:
Cluster:
{cluster}

Statement:
{statement}

Output:
Return only a JSON object with a single key "generated_query". The value for the key is the generated query for the input statement that belongs to the input cluster.

Figure 9: Prompt used for generating user queries.