Assisting Mathematical Formalization with A Learning-based Premise Retriever

Yicheng Tao[1,2,4*] Haotian Liu[1†] Shanwen Wang[2,4‡] Hongteng Xu[1,3§]
[1]Gaoling School of Artificial Intelligence, Renmin University of China
[2]School of Mathematics, Renmin University of China
[3]Beijing Key Laboratory of Big Data Management and Analysis Methods
[4]Innovation Laboratory of Mingli College, Renmin University of China
hongtengxu@ruc.edu.cn

Abstract

Premise selection is a crucial yet challenging step in mathematical formalization, especially for users with limited experience. Due to the lack of available formalization projects, existing approaches that leverage language models often suffer from data scarcity. In this work, we introduce an innovative method for training a premise retriever to support the formalization of mathematics. Our approach employs a BERT model to embed proof states and premises into a shared latent space. The retrieval model is trained within a contrastive learning framework and incorporates a domain-specific tokenizer along with a fine-grained similarity computation method. Experimental results show that our model is highly competitive compared to existing baselines, achieving strong performance while requiring fewer computational resources. Performance is further enhanced through the integration of a re-ranking module. To streamline the formalization process, we will release a search engine that enables users to query Mathlib theorems directly using proof states, significantly improving accessibility and efficiency. Codes are available at https://github.com/ruc-ai4math/Premise-Retrieval.

1 Introduction

Formalized mathematics has recently attracted significant attention. It helps verify existing mathematical results, identify errors in the literature, and has the potential to accelerate the peer-review process for mathematical papers. This process involves proving natural language theorems within a strict formal logical framework. Interactive theorem provers (ITPs), or proof assistants, are commonly used for this purpose. In recent years, mathematicians have actively participated in developing large formalized theorem libraries using ITPs such as Lean [7], Coq [2], and Isabelle [25].

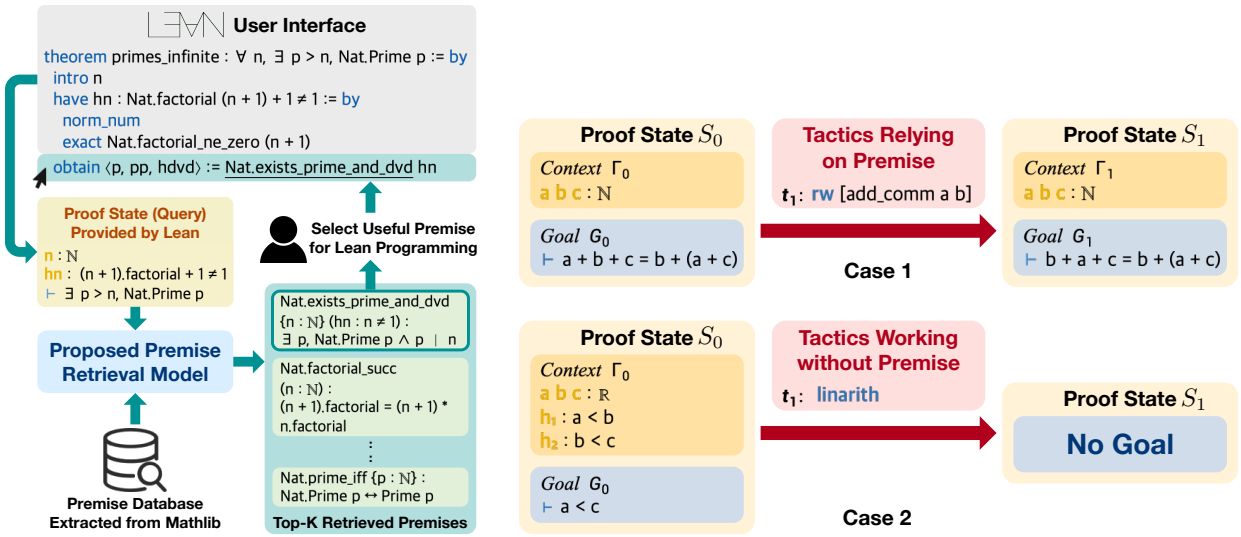
However, the process of mathematics formalization often demands extensive experience with formalization and a high level of mathematical expertise. Therefore, there has been a sustained demand for automated auxiliary tools in the formalization field, such as premise retrievers. As illustrated in Figure 1(a), premise retrieval aims to retrieve useful premises based on the current proof state provided by the formalization platform. An effective formal theorem premise retrieval

*This work was partially done when Yicheng Tao was working as a research intern at Gaoling School of Artificial Intelligence.

†Yicheng Tao and Haotian Liu have equal contributions to this work.

‡Corresponding Author

§Corresponding Author



(a) (b)

Figure 1: (a) A schematic diagram illustrating user interactions with an ITP (e.g., Lean) given our premise retrieval model. (b) Examples of the theorem-proving process in Lean.

system can help mathematicians quickly identify relevant theorems, thereby accelerating the proof process. For example, LeanDojo [41] shows that a retrieval-augmented generator achieves a higher proof pass rate compared to a standalone generator without premise selection.

The early premise retrieval methods leverage internal APIs of ITPs to identify applicable theorems through pattern-matching algorithms. These methods, however, rely on strict matching criteria and may miss relevant theorems. Recently, learning-based methods [41, 24] have been proposed to encode formalized theorems in a latent space and retrieve them by computing their embedding similarities. The learning-based methods can be further categorized based on the query language: natural language queries or formal proof state queries. Natural language-based retrieval [9] requires users to have a sufficient mathematical understanding of the current proof state, posing a significant challenge for beginners. In contrast, formal language-based retrieval is more user-friendly, but it often relies on fine-tuning pre-trained natural language models [41] because of the scarcity of formal proof data. As a result, the fundamental differences between formal and natural languages introduce a semantic gap, leading to suboptimal performance.

The above challenges motivate us to design and learn a more effective premise retrieval system that directly leverages the formal proof state to retrieve premises, improving accuracy and usability for formalized mathematics. Specifically, our premise retrieval model consists of a context-free retrieval (CFR) module and a context-aware re-ranking (CAR) module, each of which takes BERT [8] as its backbone. The CFR module derives embeddings for premises and proof states (i.e., queries) and retrieves relevant premises based on their similarity to the input proof states. The CAR module is employed to reorder the retrieved premises, and accordingly, improves the top-k recall of the retrieval results. Unlike approaches that merely fine-tune pre-trained language models, we first pre-train a BERT model from scratch based on existing formalized corpora, in which a new tokenizer is learned for formal language. Based on the pre-trained BERT, we further learn the CFR and CAR modules separately via contrastive learning [5, 13]. Experiments show that the retrieval results of our model surpass state-of-the-art models in performance while still maintaining computational efficiency. Furthermore, to evaluate the practicality of our premise retrieval model, we integrate

it with an independently trained tactic generator, creating a retrieval-augmented automated theorem prover. We then assess this system using the test dataset and MiniF2F benchmark [45], demonstrating the effectiveness of our retrieval model in facilitating the proof process.

To expedite the formalization process and provide convenience for provers, we are releasing our model, along with a web search engine available to all users. Additionally, we have developed a toolkit to facilitate the setup of local databases and the deployment of the model for those working on individual projects. Our contributions can be summarized as follows:

- We train an effective retrieval model for formalized theorems, with a new tokenizer for formal language, which achieves notable results at low computational costs and capable of running on a personal computer.
- We evaluate various retrieval models in a retrieval-augmented theorem proving pipeline, demonstrating the advantages of the proposed retrieval model in assisting proof generators.
- We deploy a theorem search engine featuring a real-time updating database, available free of charge to provers, and provide a toolkit to facilitate local model deployment for users.

2 Preliminaries and Related Work

2.1 Preliminaries in Formalization and Lean

Interactive theorem provers (ITPs) are software systems that provide a formal language for users to construct proofs verified by small, deterministic kernels. The process of encoding mathematical definitions, theorems, and proofs using this formal language is known as formalization. Modern ITPs, such as Isabelle [25], Coq [2], and Lean [7], are built on advances in type theory and have become sufficiently expressive to formalize complex, cutting-edge mathematics[3]. Among these, Lean stands out for its powerful automation tools and the extensive, community-driven mathematical library Mathlib, making it a preferred choice for mathematicians engaged in formalizing mathematical knowledge.

As a proof assistant, Lean provides a user interface for constructing formal proofs. The formalization process typically begins by translating a mathematical statement into Lean. Lean then analyzes the statement and presents the user with a proof state, which consists of a collection of hypotheses and the proposition to be proved. The user provides commands to modify the proof state until no goals remain. In this context, the set of hypotheses is referred to as the *context*, and the proposition is called the *goal*.

While there are various styles of proofwriting, machine learning researchers commonly adopt the tactic-state transformation as an abstraction of the process. A tactic is a piece of program that modifies the current proof state. Experienced Lean users can develop custom tactics through meta-programming, utilizing Lean's native automation tools. Pre-defined tactics, such as `simp`, `linarith`, and so on, have significantly enhanced the efficiency of formalization.[1]

We present a formal specification of the theorem-proving process in Lean. Starting with an initial proof state $S_0 = \{\Gamma_0, G_0\}$, where Γ_0 represents the initial context and G_0 the initial goal, the prover must provide a sequence of tactics $\{t_i\}_{i=1}^{n}$ to construct the proof. This sequence can be visualized as a linked chain of transformations, i.e.,

$$S_0 \xrightarrow{t_1} S_1 \xrightarrow{t_2} S_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} S_n. \tag{1}$$

[1]The `simp` tactic uses lemmas and hypotheses to simplify the main goal target or non-dependent hypotheses. The `linarith` tactic attempts to find a contradiction between hypotheses that are linear (in)equalities. More information can be found at `https://lean-lang.org/doc/reference/latest/Tactic-Proofs/Tactic-Reference/#tactic-ref`.

The proof is complete when G_n = "No Goals". It is important to note the varied nature of tactics. As shown in Figure 1(b), some require additional premises to take effect, while others operate independently. Furthermore, the behavior of certain tactics may vary depending on the presence of premises.

2.2 Related Work

2.2.1 Learning-based Retrieval Models

In the field of information retrieval, traditional methods like BM25 [12, 30] and TF-IDF [31] rely on term information and document statistics to match queries with relevant documents. These methods are limited in performance because they do not consider the semantic similarities between queries and documents. However, with the advancement of deep learning, researchers have developed learning-based retrieval models, often referred to as dense retrieval models [1, 16, 28, 27, 19, 20]. These models typically fall into two categories.

The first approach [16, 28] encodes queries and documents into embeddings separately. Their similarity is then computed in the embedding space, and the top-ranked documents are retrieved as the final results. A key advantage of this method is that document embeddings can be precomputed, allowing the system to embed only the query during retrieval and improving computational efficiency significantly. However, this approach lacks direct interaction between queries and documents, which may limit its retrieval performance. The second approach [27, 26] concatenates the query and document as a single input to the encoder model, which then outputs a relevance score between them. This method can capture richer interactions between queries and documents, leading to improved performance. However, it incurs higher computational costs since the concatenation must be performed for each query-document pair. Recently, researchers have increasingly embraced the paradigm of fine-tuning pre-trained models for various retrieval tasks [21, 38, 18], leading to substantial improvements in performance.

2.2.2 Mathematics Premise Selection

Previous mathematics theorem selection differs from natural language information retrieval, they often need to retrieve mathematical formulas in the library. Mathematical formulas are highly structured and are typically represented using tree-based structures [42, 10]. Retrieval methods [6, 44, 32, 17, 47, 33, 43] for such formulas rely on structural searches, which identify similarities by matching substructures across various features. There are also methods that utilize neural networks to retrieve mathematical information [22, 33, 46, 23, 29, 14].

However, premise selection in Lean is different from the aforementioned scenario. Theorems in Lean are formalized as Lean code, which lacks the rich structural information inherent to mathematical formulas. As a result, Lean's internal API[2] employs pattern matching to identify theorems applicable to the current proof state. This approach, however, enforces strict matching criteria, which often leads to failures in retrieving relevant theorems. In addition, applications like Moogle[3] and leansearch [9] have been developed to enable searches based on natural language descriptions of the target query. However, they place a higher demand on the precision of natural language formulations for both the query and the formal theorems. In contrast, premise selection based on purely current formal proof state is a more straightforward approach. LeanDojo [41] employs a learning-based retrieval model that encodes the proof state and theorems independently, subsequently calculating their similarity to perform the retrieval task. However, its performance is less

[2] `https://loogle.lean-lang.org/`

[3] `https://www.moogle.ai/`

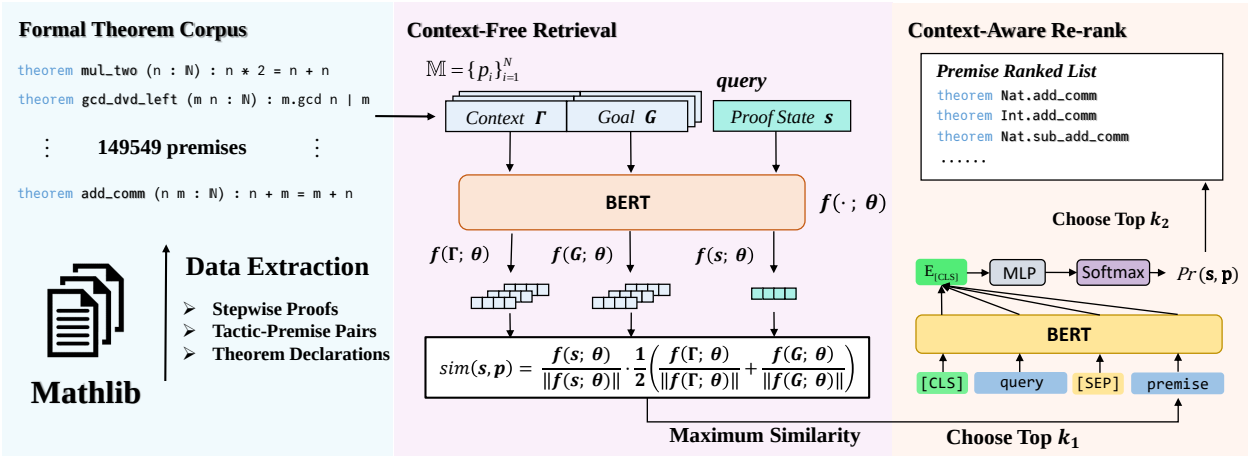


Figure 2: The overview of the retrieval framework. We extract 149,549 premises from Mathlib as our formal theorem corpus. In the context-free retrieval stage, all the premises are encoded by averaging the embeddings of context and goal. The top-k_1 premises retrieved in this stage will be re-ranked in a context-aware manner.

satisfactory, likely because they fine-tune the model pre-trained on natural language which fails to adapt well to the formal structure and syntax of Lean code.

2.2.3 AI-Assisted Formalization

Tools based on deep learning, particularly large language models (LLMs), have been developed to expedite the formalization process. These language models [41, 34, 39] assist by automating the formalization of statements, suggesting relevant premises, and generating proofs. Notably, LeanCopilot [34] was designed to enable Lean users to leverage pre-trained AI models in their formalization workflows. The Lean community is actively investigating further applications of AI across various formalization projects.

3 Proposed Method

In this section, we present the proposed retrieval framework for formal premise selection. As shown in Figure 2, this framework includes three stages:

- Extract data from the formal mathematics library to construct the theorem corpus.
- The context-free retrieval stage involves encoding the premises and proof states using a learning-based proof state encoder and calculating semantic similarities.
- In the context-aware re-ranking stage, the retrieved premises are concatenated with the query and evaluated using a cross-encoder for classification and re-ranking.

3.1 Data Preparation

3.1.1 Task Definition

Let $\mathbb{M} = \{\boldsymbol{p}_i\}_{i=1}^{N}$ denote the library to be searched, where each $\boldsymbol{p}_i$ represents a theorem within the library. Since theorems are functions that accept hypotheses as input and yield proofs of specified

propositions, $\boldsymbol{p}_i$ can be represented as $\{\Gamma_{\boldsymbol{p}_i}, G_{\boldsymbol{p}_i}\}$, where $\Gamma_{\boldsymbol{p}_i} = [v_1, \ldots, v_n]_{\boldsymbol{p}_i}$ is the argument list of the theorem and G_i is the output of the theorem. This representation is isomorphic to a proof state. Given the current proof state $\boldsymbol{s}$, the parameterized retrieval model $\Phi(\boldsymbol{s}; \mathbb{M}, \Theta)$ assigns a score to each premise and returns the top-k results. Note that a proof state may contain several cases, denoted as $\boldsymbol{s} = \{\boldsymbol{s}^i\}_{i=1}^{|\boldsymbol{s}|}$, where $\boldsymbol{s}^i = \{\Gamma^i_{\boldsymbol{s}}, G^i_{\boldsymbol{s}}\}$.

From human-written proofs, we can extract tactic steps $\mathbb{T} = \{t_i\}_{i=1}^{T}$. Each tactic step contains the proof state before and after executing the tactic, as well as the premises used in the tactic, denoted as $t_i = (\boldsymbol{s}_i^{before}, \{\boldsymbol{p}_{ij}\}_{j=1}^{n_i}, \boldsymbol{s}_i^{after})$, where $\boldsymbol{p}_{ij} \in \mathbb{M}$. We collect state-premise pairs from these tactics to form our training dataset. We consider the state before/after a tactic both relevant to the premises used in this step since the state before should satisfy the hypotheses of the premises, while the state after will contain patterns in the goal of the premises. Thus we can form our dataset as $\mathcal{D} = \{(\boldsymbol{s}_i, \mathcal{P}_i)\}_{i=1}^{|\mathcal{D}|}$, where $\boldsymbol{s}_i$ are distinct proof states and $\mathcal{P}_i$ is the set of relevant premises of $\boldsymbol{s}_i$.

3.1.2 Data Extraction

Several tools have been developed to extract data from a Lean project. To meet our requirements, we utilize the script from LeanDojo and incorporate additional features. We applied this pipeline to Mathlib[4]. The extracted information covers all the premises and proofs in this project, along with metadata such as file dependencies. For each premise, we extract its argument list and output. The proofs are lists of tactics, with the state before/after the tactics and premises used in each step. Figure 3 illustrates the statistics related to context length and the number of premises in the dataset. The context length represents the length of Γ, while the premise number indicates how many premises are utilized in a specific tactic. It's important to note that tactics that do not have any premises are excluded from these statistics.

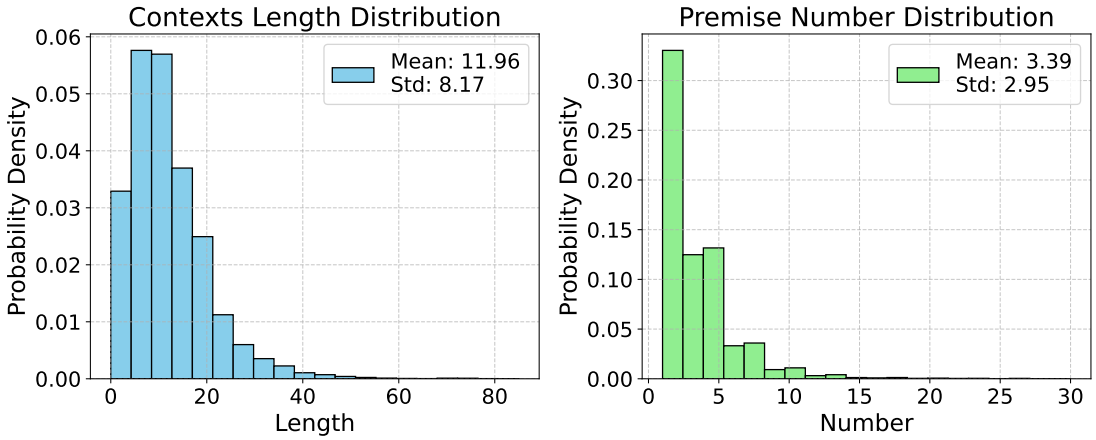


Figure 3: The probability density distribution of context lengths and premise numbers in the extracted dataset.

3.2 Model Architecture

Our premise retrieval model consists of two components: a context-free retrieval (CFR) module and a context-aware re-ranking (CAR) module. In detail, the CFR module retrieves a large set of

[4]We used tag v4.10.0 of mathlib4. `https://github.com/leanprover-community/mathlib4/tree/v4.10.0`

relevant theorems for a given proof state. Subsequently, the CAR module refines the ranking of the top-k theorems retrieved by the CFR module, further improving the accuracy of the retrieval process.

3.2.1 Context-Free Retrieval

In the context-free retrieval stage, we adopt a dense retrieval method [16]. This approach maps both the proof state and the theorem into a shared latent space, enabling retrieval through the computation of their cosine similarity. We design the model based on the BERT encoder architecture denoted as $f(\theta)$. For a given input text, we process it through the model and apply average pooling over the last hidden states to derive its latent representation, denoted as $f(\cdot;\theta)$.

We preprocess the context and goal for each state and premise to eliminate the effect of the theorem name and achieve a unified format. We prepend a special token `<VAR>` to each element of the context Γ and `<GOAL>` to the goal G, then concatenate them to form a new string. Here is an example:

Raw theorem:

```
theorem Nat.add_comm (n m : Nat) : n + m = m + n
```

After process:

```
<VAR> n m : Nat <GOAL> n + m = m + n
```

The retrieval model measures the relevance between a proof state $\boldsymbol{s}$ and a premise $\boldsymbol{p}$ by computing the similarity of their embeddings. The conventional similarity can be defined as

$$\mathrm{sim}(\boldsymbol{s},\boldsymbol{p}) = \frac{f(\boldsymbol{s};\theta)}{\|f(\boldsymbol{s};\theta)\|} \cdot \frac{f(\boldsymbol{p};\theta)}{\|f(\boldsymbol{p};\theta)\|}. \tag{2}$$

However, considering that some states may only correlate with either the arguments or the goal, we encode the premise arguments and goal separately, which can be denoted as $f(\Gamma_{\boldsymbol{p}},\theta)$ and $f(G_{\boldsymbol{p}},\theta)$. Accordingly, we design a fine-grained similarity between the given state $\boldsymbol{s}$ and premise $\boldsymbol{p}$, which can be formulated as

$$\mathrm{sim}(\boldsymbol{s};\boldsymbol{p}) = \underbrace{\frac{f(\boldsymbol{s};\theta)}{\|f(\boldsymbol{s};\theta)\|}}_{\text{state embedding}} \cdot \underbrace{\frac{1}{2}\left(\frac{f(\Gamma_{\boldsymbol{p}};\theta)}{\|f(\Gamma_{\boldsymbol{p}};\theta)\|} + \frac{f(G_{\boldsymbol{p}};\theta)}{\|f(G_{\boldsymbol{p}};\theta)\|}\right)}_{\text{premise embedding}}. \tag{3}$$

This similarity is designed based on the principle that the context of the proof state should first meet the preconditions of the available premises and, afterward, determine whether the conclusions drawn from these premises fulfill the current requirements. Compared to the conventional similarity, this fine-grained design aims to explicitly measure the similarity from both perspectives, capturing premises that can be applied in the current context, along with premises that can be a solution to the current goal.

3.2.2 Context-Aware Re-ranking

While facilitating the retrieval model and vector store enables fast and efficient queries, the lack of interaction between the state and premise may lead to lower accuracy. To address the limitations of this context-free approach, we adopt a context-aware method to re-ranking the results from

first stage. To be more specific, we design the re-ranking model based on BERT architecture that follows the sequence-pair classification of [26]. The state and premise will be concatenated together by `[SEP]` as the re-ranking module's input sequence, i.e.,

$$\text{Input} = \{[\text{CLS}],\ \boldsymbol{s},\ [\text{SEP}],\ \Gamma_{\boldsymbol{p}},\ G_{\boldsymbol{p}}\}. \tag{4}$$

The module obtains the embeddings of the input. Passing the `[CLS]` embedding through an affine projection followed by a sigmoid layer, we obtain the relevant probability of the state and premise, i.e.,

$$Pr(\boldsymbol{s}, \boldsymbol{p}) = \sigma(\mathbf{W} \cdot \mathbf{h}_{[CLS]} + \mathbf{b}), \tag{5}$$

where $\mathbf{h}_{[CLS]}$ is the embedding of the `[CLS]` token, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias term, and σ represents the sigmoid activation function. For top-k retrieval, we first use the retrieval model to find k_1 candidates. The re-ranking model serves as a filter and returns top-k_2 as the final result.

Note that, existing re-ranking methods [27, 26] improve retrieval performance by concatenating the state and each premise and passing them through their re-ranking models. This strategy has a high computational cost during inference due to the combining process of state and premise. As a trade-off, we apply a re-ranking model to refine the ranking of premises retrieved by the retrieval model.

3.3 Learning Algorithm

Based on the Masked Language Modeling (MLM) [36, 8] method, we pre-train both of the CFR and CAR modules on the formalized corpus we collected. Their tokenizers are the same and trained on formalized language corpus using WordPiece [35] algorithm. Then, we use the state-premises pairs to fine-tune the pre-trained modules separately by contrastive learning.

3.3.1 Contrastive Learning of CFR Module

For $(\boldsymbol{s}_i, \mathcal{P}_i) \in \mathcal{D}$, we use $\boldsymbol{p}_{ij} \in \mathcal{P}_i$ along with negatives sampled from $\mathbb{M}$ to construct a set $\mathcal{P}'_{ij}$, which contains one positive $\boldsymbol{p}_{ij}$ and $|\mathcal{P}'_{ij}| - 1$ negatives. The optimization objective of contrastive learning is formulated as

$$\min \sum\nolimits_{i=1}^{|\mathcal{D}|} \sum\nolimits_{j=1}^{|\mathcal{P}_i|} -\log \frac{\exp(\text{sim}(\boldsymbol{s}_i, \boldsymbol{p}_{ij})/\tau)}{\sum_{\boldsymbol{p}' \in \mathcal{P}'_{ij}} \exp(\text{sim}(\boldsymbol{s}_i, \boldsymbol{p}')/\tau)}, \tag{6}$$

where τ refers to the temperature. We use the Homogeneous In-Batch Negative Sampling, which calls for many negative samples to guarantee the embedding's discriminativenss [15, 37, 28]. In our work, this is implemented by the usage of in-batch negatives — for each query, the negatives are randomly sampled from the corpus, excluding the other positive premises of the query. Given a batch of B samples, it results in $B \times |\mathcal{P}'_{ij}| - 1$ negative samples.

3.3.2 Contrastive Learning of CAR module

The CAR module is learned in the same contrastive learning framework, in which the choice of negatives is crucial. We sample hard negatives from the top-k_1 premises selected by the retrieval model and they will be used in the re-ranking model training process. During testing, we also use the retrieval model to retrieve the top-k_2 premises and re-ranking them by the re-ranking

model. The loss function used for training the re-ranking model is the cross-entropy loss, and the optimization objective is formulated as

$$\min \sum_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{P}_i|} -\log \frac{Pr(\boldsymbol{s}_i, \boldsymbol{p}_{ij})}{\sum_{\boldsymbol{p}' \in \mathcal{P}'_{ij}} Pr(\boldsymbol{s}_i, \boldsymbol{p}')}, \tag{7}$$

where $\mathcal{P}'_{ij}$ is made up by one positive $\boldsymbol{p}_{ij}$ and several hard negatives. $Pr(\boldsymbol{s}, \boldsymbol{p})$ is the relevant probability obtained via Eq. (5).

3.3.3 Training a Tactic Generator for Evaluation

After training the retrieval model, we integrate it with the generator for testing. Most prior work on generators [39, 34, 41] utilizes state-tactic pairs to fine-tune a pre-trained model for tactic generation. Leandojo [41] incorporates premise retrieval by leveraging the retrieval model to extract relevant premises based on the state from the training dataset. The premises are then concatenated with the state to form premise-augmented state-tactic pairs used for fine-tuning. However, this approach results in a high degree of coupling between the retrieval model and the generator, as prior knowledge of the retrieval result distribution is incorporated during training.

To ensure a fair experimental setup, we aim to decouple the generator from the retrieval model and train the generator independently. As a substitution of the retrieval model, we randomly select the positive and negative premises from the library and prepend them to the state, thereby creating premise-augmented state-tactic pairs. For each tactic, we add up to ten premises. The number of positive premises selected is randomly determined, ranging from one to the lesser of the total available positive premises or ten. The remaining premises are negative. Half of these negatives are selected from the same module as the positive samples, while the remaining are chosen from other modules. Following [41], we use these pairs to fine-tune the ByT5 [40] model for tactic generation. The loss function we use is typically cross-entropy loss.

4 Experiments

To demonstrate the effectiveness of our retrieval method, we provide conducted experiments to answer the following research questions (**RQs**).

- **RQ1:** How does our proposed retrieval model compare to state-of-the-art retrieval models in formal premise retrieval?
- **RQ2:** How do our model architecture and training strategy affect the model performance?
- **RQ3:** What is the robustness of our method with respect to variations in data and model hyperparameters?
- **RQ4:** What is the impact of our retrieval model on the performance of retrieval-augmented theorem provers?

4.1 Experiment Setup

4.1.1 Implementation Details

Our retrieval and re-ranking models are based on the BERT architecture, which consists of 6 layers, each with 12 attention heads. It features a hidden size of 768 and an intermediate size of 3,072,

complemented by a vocabulary size of 30,522. For the retrieval model, the maximum position embeddings are configured at 512. The maximum length for states is set at 512 and for premises' arguments and goals set at 256 respectively. The batch size is set at 32 and $|\mathcal{P}'_{ij}|$ mentioned in 3.3.1 is set at 2. In contrast, the re-ranking model has its maximum position embeddings set at 1,024. We set the batch size at 2, gradient accumulation steps at 8, and $|\mathcal{P}'_{ij}|$ at 8. We construct the pre-training corpus by concatenating states from the training set and all premises' argument lists and goals from the corpus. The results of our method presented in Table 3 are obtained by first training the retrieval model and selecting the top-100 results as hard negative candidates. We then train the re-ranking model and use it to reorder the top-20 results from the retrieval stage. The experiments are conducted on 8 RTX 4080 servers.

4.1.2 Baselines

As mentioned above, ReProver [41] is a model specifically trained on the Lean dataset, using formal states and premises. Therefore, it serves as our primary baseline. Moreover, we retrain the model using our dataset and following the setting in [41]. In addition to ReProver, we compare against several other commonly used retrieval models that have demonstrated strong performance in their specific domains.

- UniXcoder-base [11]: It is one of the state-of-the-art code embedding models to transform code snippets into hidden vectors. It leverages multimodal data, such as code comments and abstract syntax trees (ASTs), to pre-train code representations.
- E5-large-v2 [37]: It is trained in a contrastive manner using weak supervision signals derived from a curated, large-scale text-pair dataset (referred to as CCPairs) and demonstrates strong performance on several English retrieval tasks.
- BGE-m3 [4]: It is distinguished for its versatility and excels in multiple functionalities, multilingual capabilities, and fine granularity retrieval tasks.

We fine-tune the three aforementioned baselines using our dataset. Following the parameters provided in [4], we set the learning rate to 1e-5 and use a linear scheduler. Due to server constraints, the batch sizes for the three models are set to 16, 8, and 6, respectively.

4.1.3 Metrics

To evaluate the performance of various retrieval models, we utilize four widely adopted metrics: Precision, Recall, F1-Score, and nDCG. Since the nDCG considers the retrieved results' relevance and position, it allows for multiple relevance levels. We define the relevance criteria as shown in Table 1, which provides the relevance scores required for calculating nDCG.

4.1.4 Data Split

To evaluate the performance of the model across different feature datasets, we employ four distinct data split strategies. In particular, we represent a proof as $P = \{t_i\}_{i=1}^{|P|}$, which is a sequence of tactics, and apply *Proof Length* and *Premise Frequency* to characterize the proof, i.e.,

$$\text{ProofLength} = |P|,\ \text{PremiseFreq} = \frac{1}{|P|}\sum\nolimits_{i=1}^{|P|} \text{PremiseNum}(t_i).$$

where $\text{PremiseNum}(t_i)$ denotes the number of external premise in a tactic. *Proof Length* reflects the complexity of the proof, while the *Premise Frequency* indicates the degree of reliance on external

Table 1: Relevance criteria for the retrieval results.

Rating	Score	Description
Match	1	Exact match to one of the premises used in the tactic.
Relevant	0.3	Within the same module with any of the premises used in the tactic.
Irrelevant	0	Situations other than the two mentioned above.

theorems during the proof process. Then, we apply the following four data split strategies to obtain four datasets:

- **Random (RD):** Randomly split the dataset.
- **Reference Isolated (RI):** Premises in the validation and test sets will not appear in the training set.
- **Proof Length (PL):** Sample proofs for the validation and test sets weighted by *Proof Length*.
- **Premise Frequency (PF):** Sample proofs for the validation and test sets weighted by *Premise Frequency*.

For each dataset, the training set contains 65,567 theorems, while the validation and test sets each consist of 2,000 theorems. The average proof length and premise frequency for each dataset are shown in Table 2.

Table 2: The average proof length and premise frequency of four different split strategies.

Split	Proof Length			Premise Frequency		
	train	valid	test	train	valid	test
RD	2.48	2.54	2.44	2.25	2.31	2.15
RI	2.41	3.54	3.62	2.20	3.03	3.16
PL	2.35	4.50	4.77	2.26	2.12	2.11
PF	2.49	2.31	2.31	2.13	4.08	4.63

4.2 Main Results (RQ1)

After training different models on the training dataset and evaluating them on the corresponding test sets, we obtain the results presented in Table 3. From this table, we can observe that our method demonstrates exceptional performance, significantly outperforming other baseline models. Specifically, UniXcoder-base [11], E5-large-v2 [15], and BGE-m3 [4] exhibit generally poor performance, which can be attributed to the substantial discrepancy between the pre-training corpora used by these models (based on natural language data) and the Lean language. Although these models are fine-tuned, they cannot fully capture the core aspects of the Lean language due to Lean's

Table 3: Premise selection results of our method and other baselines on different data splits.

Split	Method	Recall			Precision			F1-score			nDCG		
		R@1	R@5	R@10	P@1	P@5	P@10	F@1	F@5	F@10	n@1	n@5	n@10
RD	UniXcoder-base	4.23	19.47	27.01	6.68	6.90	5.04	5.18	10.19	8.50	0.2283	0.3455	0.3958
	E5-large-v2	4.42	18.99	27.29	6.90	6.75	5.09	5.39	9.96	8.58	0.2313	0.3458	0.3952
	BGE-m3	4.00	18.56	25.29	6.40	6.51	4.64	4.92	9.64	7.84	0.2374	0.3481	0.3930
	ReProver	11.79	28.78	36.69	19.80	10.90	7.23	14.78	15.81	12.08	0.3351	0.4072	0.4617
	Ours	**15.17**	**38.20**	**46.53**	**26.80**	**14.87**	**9.30**	**19.38**	**21.41**	**15.51**	**0.3731**	**0.4698**	**0.5163**
RI	UniXcoder-base	4.54	17.55	24.16	6.68	6.18	4.54	5.41	9.15	7.64	0.2303	0.3258	0.3768
	E5-large-v2	4.80	18.21	24.73	7.42	6.55	4.69	5.83	9.64	7.88	0.2356	0.3407	0.3878
	BGE-m3	4.52	17.69	23.76	7.05	6.19	4.45	5.51	9.17	7.49	0.2306	0.3312	0.3788
	ReProver	5.05	14.26	19.48	9.30	5.77	4.10	6.54	8.22	6.77	0.2442	0.3059	0.3563
	Ours	**7.79**	**23.38**	**30.91**	**14.76**	**9.30**	**6.38**	**10.20**	**13.31**	**10.58**	**0.2731**	**0.3736**	**0.4322**
PL	UniXcoder-base	3.94	15.45	22.72	6.70	5.66	4.32	4.96	8.29	7.25	0.1958	0.2804	0.3272
	E5-large-v2	3.66	14.65	21.65	6.25	5.47	4.23	4.61	7.96	7.08	0.1849	0.2735	0.3229
	BGE-m3	3.99	14.65	20.53	7.06	5.41	3.92	5.10	7.90	6.59	0.1974	0.2781	0.3218
	ReProver	11.16	26.83	33.96	19.64	10.31	6.83	14.23	14.90	11.38	0.3059	0.3699	0.4232
	Ours	**14.39**	**34.99**	**41.61**	**25.18**	**13.80**	**8.45**	**18.31**	**19.80**	**14.04**	**0.3479**	**0.4319**	**0.4781**
PF	UniXcoder-base	2.69	15.03	21.91	5.19	6.92	5.43	3.54	9.48	8.70	0.2251	0.3426	0.3973
	E5-large-v2	3.08	14.44	22.11	5.84	6.70	5.39	4.04	9.16	8.67	0.2300	0.3464	0.4023
	BGE-m3	2.59	13.85	20.05	5.42	6.27	4.86	3.51	8.63	7.83	0.2320	0.3404	0.3910
	ReProver	8.29	22.74	30.21	18.99	11.83	8.30	11.54	15.57	13.02	0.3257	0.3966	0.4544
	Ours	**11.44**	**31.02**	**38.88**	**26.38**	**15.96**	**10.52**	**15.96**	**21.08**	**16.56**	**0.3764**	**0.4693**	**0.5238**

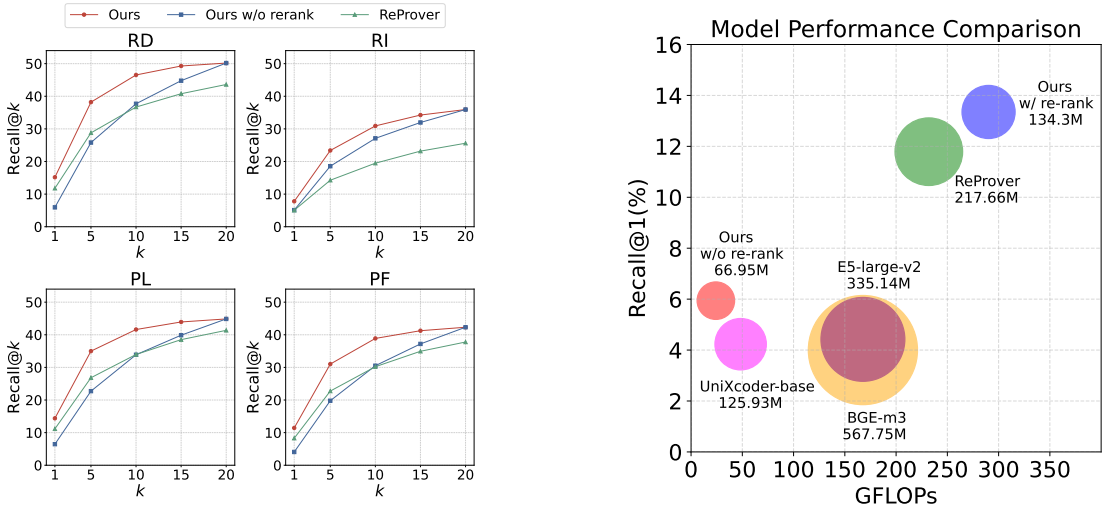
Table 4: Ablation studies on pre-training, tokenizer, and similarity calculation on Random split dataset for the CFR module.

Method			Recall			Precision			F1-score			nDCG		
Pre-train	Tokenizer	Similarity	R@1	R@5	R@10	P@1	P@5	P@10	F@1	F@5	F@10	n@1	n@5	n@10
×	×	fine-grained in Eq. (3)	4.63	20.09	29.05	8.35	7.45	5.62	5.96	10.87	9.42	0.2443	0.3503	0.4030
×	✓	fine-grained in Eq. (3)	4.93	20.49	30.33	8.04	7.50	5.79	6.11	10.98	9.72	0.2368	0.3493	0.4034
✓	×	fine-grained in Eq. (3)	6.31	23.81	34.21	10.34	8.77	6.59	7.84	12.82	11.05	0.2484	0.3655	0.4240
✓	✓	conventional in Eq. (2)	5.65	25.87	36.78	9.36	9.55	7.12	7.05	13.95	11.92	0.2382	0.3662	0.4238
✓	✓	fine-grained in Eq. (3)	5.94	25.81	37.70	9.80	9.53	7.24	7.40	13.92	12.15	0.2422	0.3674	0.4293

unique syntax and semantic features and thus suffer suboptimal performance. As a result, these models likely require more specialized fine-tuning strategies and parameters tailored to the Lean language to improve their performance.

In contrast, the ReProver [41] model performs relatively well, likely due to its specifically designed parameter-tuning strategy. Furthermore, ReProver utilizes ByT5 [40], a byte-level tokenizer, which may allow the model to better handle the special symbols and structures in the Lean language. Our approach, on the other hand, improves performance by retraining the tokenizer on the Lean corpus and pre-training the retrieval model, allowing it to better adapt to the syntax and semantic features of Lean, resulting in a significant boost in overall performance.

For different data splitting strategies, we observe that random data splitting is relatively easy. However, when the theorems in the test set involve longer proofs or require a larger number of premises, the performance of the model tends to be lower. The reference isolated method, where the premises in the test set have not appeared during training, is the most challenging. It requires the model to exhibit strong generalization ability. Nevertheless, under each of these splitting strategies, our model consistently outperforms other baseline methods, which indicates that our model exhibits superior adaptability.



(a) (b)

Figure 4: (a) The Recall@k of our method with or without re-ranking, compared with ReProver. (b) The performance and efficiency comparison on Random split datasets. The size of each point reflects the parameter size of each model.

4.3 Ablation Studies (RQ2)

4.3.1 Impacts of Pre-Training, Tokenizer, and Similarity Calculation

We evaluate the impacts of pre-training the model, training the specific tokenizer for Lean, and the proposed similarity calculation in Table 4, respectively. To ensure a fair comparison, we filter out the impact of the re-ranking module and only compare the results provided by the retrieval module. This table shows a significant performance improvement when the model is pre-trained on the Lean corpus compared to not performing pre-training. This indicates that the pre-training process helps the model acquire semantic information about Lean in advance. The experiment also shows that if we only pre-train the model without training a new tokenizer, the performance when k=1 is slightly better but degrades a lot when k=5 or 10. This is because Lean has a formal language system, which differs from natural languages, highlighting the necessity of training a dedicated tokenizer for Lean. In addition, we evaluate the impacts of different similarity calculation methods on the retrieval results. The results in Table 4 emphasize the benefits of assessing similarity in a fine-grained manner.

4.3.2 The Effect of Re-ranking Model

We validate the effectiveness of using the proposed re-ranking module. As shown in Figure 4 (a), when only the CFR module is used, our model outperforms the ReProver only for larger values of k in Recall@k (e.g., $k \geq 10$). Applying the CAR module improves the recall of our model when $k < 10$, so that our model can consistently outperform ReProver and users are likely to find useful premises in a relatively short list. In other words, re-ranking contributes to a significant improvement in performance. Figure 4 (b) presents a comparison for our model and other baselines in terms of model size and inference GFLOPs. Without re-ranking, our model achieves the lowest GFLOPs and it outperforms in Recall@1 most of the baselines except ReProver. When re-ranking the top-5 retrieval results by our CAR module, our model outperforms all the baselines and its

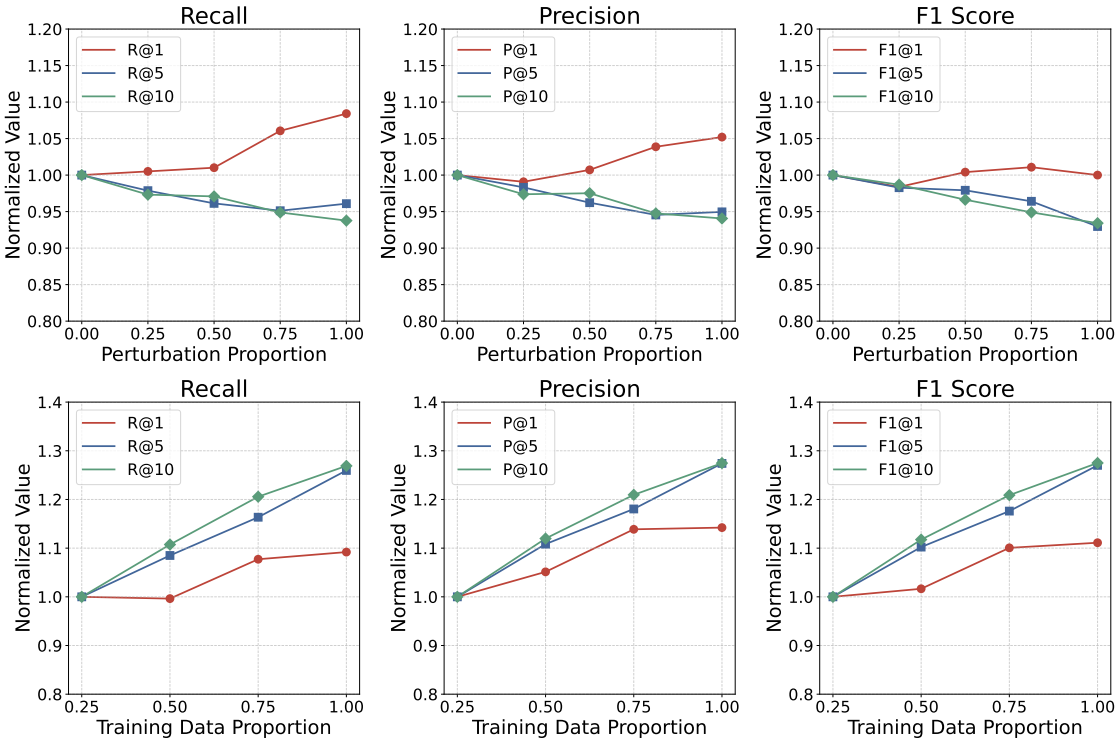


Figure 5: The variation of the normalized values of the model's metrics under different data perturbations.

GFLOPs and model size are comparable to those of ReProver.

In principle, the CAR module helps improve the Recall of our model, but its complexity increases linearly with the number of results to re-rank. Therefore, the CAR module of our model is optional in practice, as there exists a trade-off between effectiveness and efficiency. The decision to re-ranking can be made based on factors such as sensitivity to the order of retrieval results or limitations on the information window. According to experimental results, when the number of retrieved results exceeds 50, the performance improvements obtained by re-ranking are limited, making it unnecessary. For user groups that are insensitive to the order of retrieval results, such as human theorem provers, it may be unnecessary to allocate additional computational resources for re-ranking. However, in the case of retrieval modules used in automated theorem provers, where high retrieval precision is required and computational cost is not a bottleneck, incorporating a re-ranking process can be justified. Note that, as a general retrieval technique, we can also apply the re-ranking procedure to other baselines. However, with larger backbone models (e.g., ReProver), they will need more computation, making it difficult to deploy for our search engine. How to further improve model performance and efficiency simultaneously is left as our future work.

4.4 Robustness Experiments (RQ3)

4.4.1 Effect of Data Perturbation

Considering the generally high quality of proofs in Mathlib, models trained on this dataset may be sensitive to the common issues of redundant variables, missing conditions, and disordered sequences in the context of proof states, resulting in poor generalization ability. We evaluate our retrieval model's robustness to low-quality inputs by perturbing the query states in the test set. Specifically,

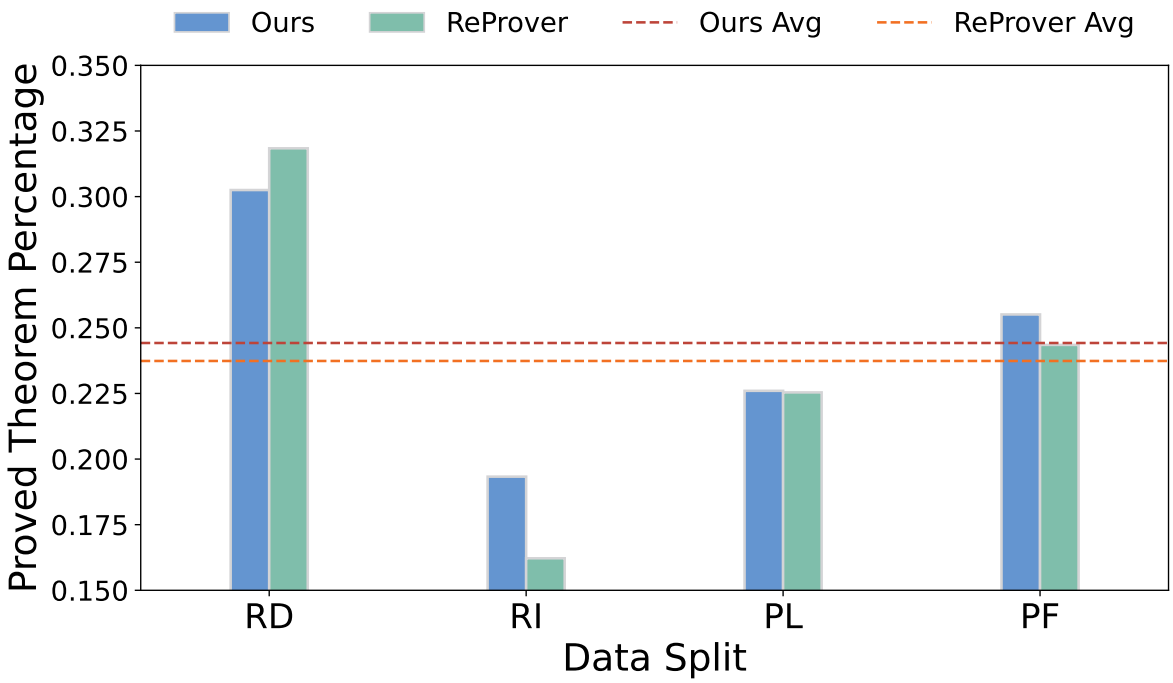


Figure 6: Retrieval-augmented theorem proving results.

we applied two perturbation strategies to a subset of the test data: shuffling the context or randomly removing 20% of the context from states with a context length of 15 or more. As shown in Figure 5, all the metrics exhibit a moderate decline as the perturbation ratio increases, with a maximum drop of approximately 6% when k=5,10 compared to the unperturbed case. Especially, the model's performance even improves when k=1. These results demonstrate that our retrieval model is robust to variations in the order of query states and the potential absence of certain local hypotheses.

4.4.2 Impact of Reduced Training Data

The sparsity of data is an inherent challenge in formalization tasks. We test the sensitivity of our retrieval model to the amount of training data. The tests are conducted on the Random split dataset. The results in Figures 5 show that the model trained on the full dataset outperforms the model trained on only 25% of the data by approximately 25% in terms of k=5,10. The consistent performance improvement suggests that there is still potential for further enhancement by increasing the amount of training data. When k=1, the performance seems to reach a bottleneck as the data size increases. Therefore, it may be necessary to incorporate re-ranking or explore alternative methods to further improve performance.

4.5 Theorem Proving Experiments (RQ4)

As mentioned in Section 3.3.3, we fine-tune ByT5 [40] and obtain a tactic generator. Facilitating the generator, we conduct theorem-proving experiments with retrieval. The results in Figure 6 show that assisted by our model, the generator averagely performs better than ReProver [41]. Additionally, we evaluate our method on the Minif2f benchmark [45] using the model trained on the random split dataset. Our approach achieves a pass@1 rate of 30.74%, while ReProver reached 28.28%, indicating that our method is more effective. This means that retrieval-augmented language

models are effective: For tasks such as formal theorem proving, which relies on external theorem libraries, an effective retrieval model is expected to enhance generation performance.

On test sets with high premise frequency but relatively short proof, our model performs better than ReProver, demonstrating the advantages of a more powerful retriever. In contrast, on test sets with long proof but low premise frequency, our model performs almost on par with ReProver. However, on the RD test sets, our model performs slightly worse than ReProver, despite the improvement in retrieval accuracy. We attribute this outcome to several factors. First, incorporating retrieved premises into the prompt increases the length of the context that the model must process compared to providing only the proof state. This imposes certain requirements on the model's parameter size and architecture, as smaller models may struggle to accurately identify useful information within longer contexts. We will explore this issue in our future work. Additionally, since Lean users are continuously developing new automated tactics, solving simpler problems often depends more on selecting the appropriate tactic than on searching for relevant theorems, making tactic selection a more efficient approach in such cases.

5 Conclusion

In this paper, we propose an innovative method that leverages data extracted from Mathlib4 to train a retrieval model specifically designed for Lean premise retrieval. Experiments show that our method outperforms previous models and demonstrates the potential of domain-specific smaller models in data-sparse tasks, such as formal premise retrieval. As shown in Figure 7, we have developed a search engine for premise retrieval based on our model[5]. We hope that our work can help accelerate mathematical formalization and contributes to the researchers in the community.

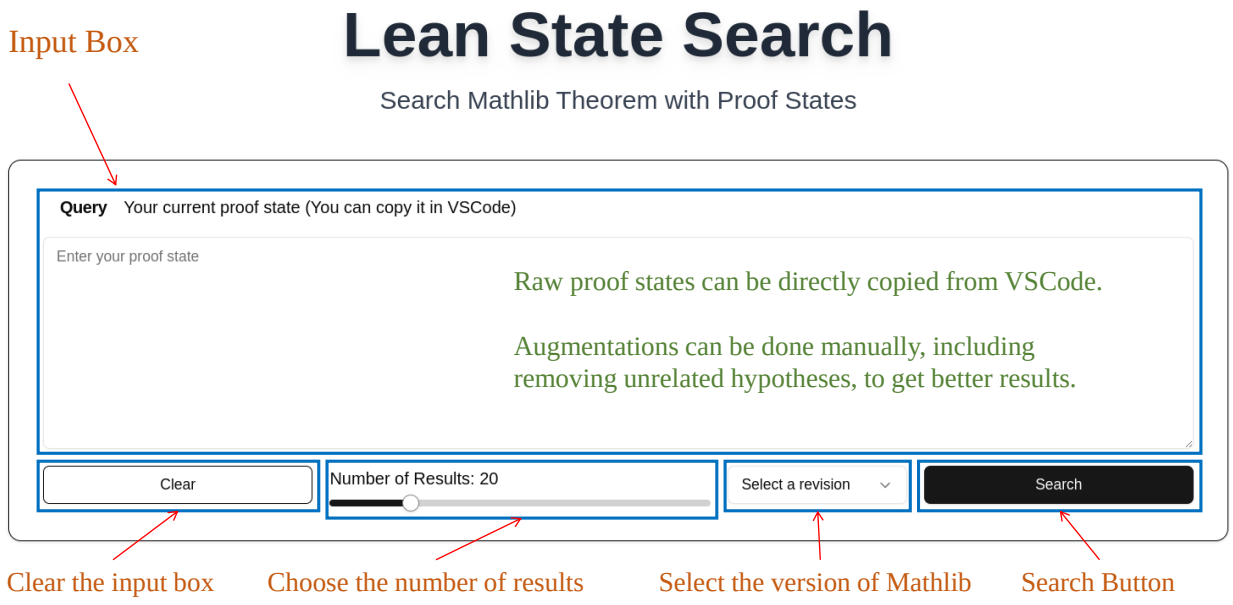


Figure 7: The interface of our premise retrieval engine.

Although our method has yielded promising results, there is still room for optimization in the retriever's backbone model. For the Lean premise retrieval task, the data corpus contains valuable semantic information, such as the hierarchy of Lean's type system, that has yet to be fully explored.

[5]We will make the search engine publicly available in the near future.

In the future, we will focus on leveraging this untapped semantic information more effectively and developing better formal language models.

References

[1] Wasi Uddin Ahmad, Kai-Wei Chang, and Hongning Wang. Context attentive document ranking and query suggestion. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 385–394, 2019.

[2] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. *The Coq proof assistant reference manual: Version 6.1*. PhD thesis, Inria, 1997.

[3] Kevin Buzzard. Mathematical reasoning and the computer. *Bulletin of the American Mathematical Society*, 61(2):211–224, February 2024.

[4] Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[6] Kenny Davila. Appearance-based retrieval of mathematical notation in documents and lecture videos. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1165–1165, 2016.

[7] Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. 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*, pages 378–388. Springer, 2015.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[9] 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, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8001–8013, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[10] Liangcai Gao, Ke Yuan, Yuehan Wang, Zhuoren Jiang, and Zhi Tang. The math retrieval system of icst for ntcir-12 mathir task. In *NTCIR*, 2016.

[11] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. UniXcoder: Unified cross-modal pre-training for code representation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[12] Donna K Harman. *Overview of the third text retrieval conference (TREC-3)*. Number 500. DIANE Publishing, 1995.

[13] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[14] Zhenya Huang, Qi Liu, Weibo Gao, Jinze Wu, Yu Yin, Hao Wang, and Enhong Chen. Neural mathematical solver with enhanced formula structure. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1729–1732, New York, NY, USA, 2020. Association for Computing Machinery.

[15] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022.

[16] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics.

[17] Giovanni Yoko Kristianto, Goran Topic, and Akiko Aizawa. Mcat math retrieval system for ntcir-12 mathir task. In *NTCIR*, 2016.

[18] Chaofan Li, Zheng Liu, Shitao Xiao, Yingxia Shao, and Defu Lian. Llama2vec: Unsupervised adaptation of large language models for dense retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3490–3500, 2024.

[19] Sheng-Chieh Lin, Minghan Li, and Jimmy Lin. Aggretriever: A simple approach to aggregate textual representations for robust dense passage retrieval. *Transactions of the Association for Computational Linguistics*, 11:436–452, 2023.

[20] Guangyuan Ma, Xing Wu, Peng Wang, Zijia Lin, and Songlin Hu. Pre-training with large language model-based document expansion for dense passage retrieval. *arXiv preprint arXiv:2308.08285*, 2023.

[21] Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2421–2425, 2024.

[22] Behrooz Mansouri and Reihaneh Maarefdoust. Using large language models for math information retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2693–2697, New York, NY, USA, 2024. Association for Computing Machinery.

[23] Behrooz Mansouri, Richard Zanibbi, and Douglas W. Oard. Learning to rank for mathematical formula retrieval. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 952–961, New York, NY, USA, 2021. Association for Computing Machinery.

[24] Maciej Mikuła, Szymon Antoniak, Szymon Tworkowski, Bartosz Piotrowski, Albert Jiang, Jin Peng Zhou, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. Magnushammer: A transformer-based approach to premise selection. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023.

[25] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

[26] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *CoRR*, abs/1901.04085, 2019.

[27] Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Understanding the behaviors of bert in ranking. *arXiv preprint arXiv:1904.07531*, 2019.

[28] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5835–5847, Online, June 2021. Association for Computational Linguistics.

[29] Anja Reusch, Maik Thiele, and Wolfgang Lehner. An albert-based similarity measure for mathematical answer retrieval. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 1593–1597, New York, NY, USA, 2021. Association for Computing Machinery.

[30] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[31] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[32] Thomas Schellenberg, Bo Yuan, and Richard Zanibbi. Layout-based substitution tree indexing and retrieval for mathematical expressions. In *Document Recognition and Retrieval XIX*, volume 8297, pages 126–133. SPIE, 2012.

[33] Moritz Schubotz, Alexey Grigorev, Marcus Leich, Howard S. Cohl, Norman Meuschke, Bela Gipp, Abdou S. Youssef, and Volker Markl. Semantification of identifiers in mathematics for better math information retrieval. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 135–144, New York, NY, USA, 2016. Association for Computing Machinery.

[34] Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Towards large language models as copilots for theorem proving in lean. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023.

[35] Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. Fast wordpiece tokenization. In *EMNLP (1)*, pages 2089–2103, 2021.

[36] Wilson L Taylor. "cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

[37] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

[38] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[39] Sean Welleck and Rahul Saha. llmstep: LLM proofstep suggestions in lean. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023.

[40] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.

[41] 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, 2024.

[42] Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15:331–357, 2012.

[43] Richard Zanibbi, Kenny Davila, Andrew Kane, and Frank Wm. Tompa. Multi-stage math formula search: Using appearance-based similarity metrics at scale. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 145–154, New York, NY, USA, 2016. Association for Computing Machinery.

[44] Jin Zhao, Min-Yen Kan, and Yin Leng Theng. Math information retrieval: user requirements and prototype implementation. In *Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*, pages 187–196, 2008.

[45] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022.

[46] Wei Zhong, Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. One blade for one purpose: Advancing math information retrieval using hybrid search. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 141–151, New York, NY, USA, 2023. Association for Computing Machinery.

[47] Wei Zhong, Shaurya Rohatgi, Jian Wu, C Lee Giles, and Richard Zanibbi. Accelerating substructure similarity search for formula retrieval. In *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part I 42*, pages 714–727. Springer, 2020.