ProofFusion: Improving Neural Theorem Proving via Adaptive Retrieval-Augmented Reasoning

MANQING ZHANG, Northwestern Polytechnical University, China
YUNWEI DONG*, Northwestern Polytechnical University, China
LINGRU ZHOU, Northwestern Polytechnical University, China
BINGXU XIAO, Northwestern Polytechnical University, China
YEPANG LIU†, Southern University of Science and Technology, China

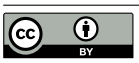
Interactive theorem proving (ITP) is a powerful approach to ensuring the correctness of complex software systems. However, it often requires substantial manual effort, which makes it costly to use in practice. Recently, neural network based approaches have shown promise in automatically generating proof tactics. Nevertheless, existing methods suffer from a long-tailed distribution in tactic usage within the training data. A few frequent tactics dominate the probability distribution, while many rare yet crucial ones are consistently suppressed in the model's candidate ranking. This distributional bias can cause potentially provable goals to be prematurely abandoned during proof search. In addition, the decision making process of neural networks when generating tactics lacks explicit reasoning traces, making it difficult for humans to explain or verify the underlying logic. To address these limitations, we propose ProofFusion, an adaptive retrieval-augmented reasoning framework that improves the proving capability of neural theorem provers without requiring retraining. Our key insight is inspired by the way human provers tackle a new theorem by consulting similar previously proven theorems to guide their own reasoning. Specifically, we develop a proof semantic-aware retriever that searches a knowledge base for semantically similar historical proof goals together with their tactic, producing a traceable set of reference decisions. We then employ a dual-track reranking fusion mechanism to integrate both the original predictions of the neural model and the retrieved reference tactics. Furthermore, to mitigate potential noise introduced by retrieval, we design a capability-adaptive retrieval mechanism that dynamically determines when retrieval should be applied. We conduct a systematic evaluation on 10,782 theorems from 26 Coq projects in a real ITP environment. Experimental results show that ProofFusion increases the number of theorems proved by four state-of-the-art neural theorem provers by an average of 6.89%, and and additionally proves 17.50% of previously unprovable theorems. In addition, it substantially improves the explainability of proof steps, achieving an average explainable proof goal proportion of 82.1% across the four provers. Together, these results demonstrate that ProofFusion is a practical and effective complement to existing neural theorem proving systems, enhancing both performance and explainability.

CCS Concepts: • **Software and its engineering** → *Formal software verification.*

Additional Key Words and Phrases: Neural Theorem Proving, Long-tailed Problem, Explainability, Adaptive Retrieval-Augmented Reasoning

*Corresponding author.
†Yepang Liu is affiliated with the Research Institute of Trustworthy Autonomous Systems and Department of Computer Science and Engineering at Southern University of Science and Technology.

Authors' Contact Information: Manqing Zhang, Northwestern Polytechnical University, Xi'an, China, zmqgeek@mail.nwpu.edu.cn; Yunwei Dong, Northwestern Polytechnical University, Xi'an, China, yunweidong@nwpu.edu.cn; Lingru Zhou, Northwestern Polytechnical University, Xi'an, China, lingruzhou@mail.nwpu.edu.cn; Bingxu Xiao, Northwestern Polytechnical University, Xi'an, China, bingxuxiao@mail.nwpu.edu.cn; Yepang Liu, Southern University of Science and Technology, Shenzhen, China, liuyp1@sustech.edu.cn.



ACM 2994-970X/2026/7-ARTFSE022
https://doi.org/10.1145/3797139

ACM Reference Format:
Manqing Zhang, Yunwei Dong, Lingru Zhou, Bingxu Xiao, and Yepang Liu. 2026. ProofFusion: Improving Neural Theorem Proving via Adaptive Retrieval-Augmented Reasoning. *Proc. ACM Softw. Eng.* 3, FSE, Article FSE022 (July 2026), 24 pages. https://doi.org/10.1145/3797139

1 Introduction

Interactive Theorem Proving (ITP) has emerged as a crucial methodology for ensuring the reliability of safety-critical systems [45, 47], including aerospace systems and medical device software. In ITP, system behaviors and properties are rigorously formalized in a logical language and verified with the assistance of proof assistants [17]. Compared with automated theorem proving [15] and model checking [6], ITP is based on higher-order logic, which enables the specification and verification of more expressive and complex system properties. This capability has facilitated its successful application to large-scale verification projects, such as the seL4 microkernel [28] and the CompCert compiler [29]. Proof development in ITP proceeds as a sequence of tactics [7], where each tactic represents a strategy or operation applied to the current proof goal. Tactics direct the proof assistant (e.g., Coq [21]) in advancing the proof by decomposing compound propositions, introducing hypotheses, or invoking previously established lemmas. By iteratively applying tactics, a complete proof script can be incrementally constructed. Despite this systematic process, proof development in ITP remains highly labor-intensive, demanding substantial expertise and time [55]. For instance, the formal verification of the CompCert compiler required approximately six person-years of effort and resulted in more than 100,000 lines of Coq proof scripts [30]. Consequently, reducing manual effort and improving the degree of automation remain fundamental research challenges in formal verification.

Existing Efforts and Limitations. Recently, researchers have introduced neural network–based approaches [2, 5, 12, 13, 51–53, 65] for automatically generating proof tactics, collectively referred to as Neural Theorem Proving. These approaches leverage large-scale, human-verified proof corpora to learn the distribution of tactics and subsequently predict the most suitable tactic for a given proof state, thereby mitigating the manual effort required in ITP. Neural Theorem Proving typically adopts a stepwise generation paradigm [37, 44], in which the model predicts one tactic at a time while search algorithms guide the exploration of possible proof paths. Specifically, from an initial proof state, the neural model generates candidate tactics, and a search procedure selects and applies the most promising option, thereby updating the proof state. This process is repeated iteratively, progressively constructing a complete proof script until the target theorem is constructed. By tightly coupling neural prediction with search, Neural Theorem Proving not only accelerates proof development but also reduces dependence on expert intervention.

Nevertheless, Neural Theorem Proving faces inherent limitations. ***[Limitation 1] Long-tailed bias in tactic learning.*** High-frequency tactics dominate the training data, while many low-frequency but critical tactics are consistently underrepresented in the model's output ranking. This distribution bias may cause some theoretically provable goals to be prematurely abandoned during the search process, thereby reducing the success rate of Neural Theorem Proving. The effect is particularly pronounced for theorems with complex structures or those relying on rare tactics. ***[Limitation 2] Lack of explainability.*** The tactic sequences generated by neural models often lack a clear logical reasoning trajectory, making it difficult for humans to understand or verify the underlying rationale. This opacity also increases the difficulty of debugging and hinders the effective integration of human expertise, as practitioners cannot easily determine why certain tactics were chosen over others.

Insights and Challenges. To address the above limitations, we draw inspiration from human theorem-proving behavior. When constructing new proofs, provers often revisit prior experiences,

consulting previously proven theorems and the tactic applied to guide their reasoning. This observation suggests that combining neural predictions with semantically relevant historical proofs could not only alleviate the long-tailed distribution problem but also enhance the explainability of generated tactics. Specifically, retrieved examples provide clear provenance for the current generation step, while their associated tactics serve as empirical priors that help rerank the model's candidate tactics, thereby mitigating the probability bias caused by the long-tail distribution. However, putting this idea into practice requires overcoming three key challenges. ***[Challenge 1] Retrieving tactic-aligned proof goals is challenging.*** Existing retrieval methods [57] mainly rely on shallow semantic similarity [11] to locate historical proofs. However, shallow semantically similar goals do not necessarily align at the tactic level, limiting the effectiveness of retrieval for tactic guidance. ***[Challenge 2] Designing an explainable fusion mechanism between retrieval and generation is challenging.*** Simply embedding retrieved goals together with the current goal may introduce noise, interfere with model decisions, and even degrade predictive accuracy. Therefore, a robust fusion mechanism is needed to effectively combine neural predictions with retrieval signals, improving the reliability and robustness of tactic selection. ***[Challenge 3] Adaptive evaluation of retrieved knowledge is challenging.*** Not all proof states are suitable for leveraging historical proofs. Without a selection mechanism, irrelevant or misleading information may be introduced, reducing search efficiency and potentially harming correctness. Hence, a dynamic evaluation method is required to exploit retrieval results only when appropriate, thereby improving the overall efficiency and reliability of the proof process.

Our Solution. We present ProofFusion, a general adaptive retrieval-augmented reasoning framework. Its core idea is to integrate neural model predictions with the semantic knowledge of historical proofs without retraining the original models, thereby improving both reasoning capacity and explainability. ProofFusion comprises three key components: (1) ***Proof Semantic-aware Retriever (Addressing Challenge 1).*** We construct a tactic-aligned dataset of semantically similar proofs and train a semantic-aware Dense Passage Retriever (DPR) [23] via contrastive learning [26]. This module retrieves both semantically similar proof data and their tactic, ensuring effective alignment at the tactic level. (2) ***Dual-track Reranking Fusion Mechanism (Addressing Challenge 2).*** At the output level of neural theorem provers, we employ a fusion mechanism that combines tactic syntax constraints with type-consistency reranking. This approach mitigates potential retrieval noise and enhances the explainability of tactic selection. (3) ***Capability-adaptive Retrieval Classifier (Addressing Challenge 3).*** Based on a neural theorem prover's ability to solve different theorems, proof goals are automatically classified as "easy" or "hard," and a classifier dynamically determines whether to incorporate retrieval information. This mechanism balances reasoning accuracy and computational efficiency. We conduct a systematic evaluation on 10,782 theorems from 26 Coq projects across four state-of-the-art neural theorem proving systems in a real ITP environment. The results show that ProofFusion improves the average proof success rate by 6.89% and additionally solves 17.50% of previously unprovable theorems, demonstrating its effectiveness in enhancing performance. It also substantially improves the explainability of proof steps across all four neural theorem provers, yielding explainable proof goal proportions in the range of 79.46–83.37%.

In summary, we make the following contributions:

- We uncover key limitations in existing neural theorem proving methods, namely the long-tailed distribution bias of tactics and the lack of explainability in generated tactics.
- We propose ProofFusion, an adaptive retrieval-augmented reasoning framework that integrates neural model predictions with the semantic knowledge of historical proofs without retraining the original models, improving both proof performance and explainability.

- We conduct a systematic evaluation on 10,782 theorems from 26 Coq projects across four neural theorem proving systems. Experiments demonstrate that ProofFusion improves proof success rates, resolves previously unprovable theorems, and enhances explainability by enabling traceable proof steps across four provers.

2 Background and Motivation

This section first introduces the workflow of search-based neural theorem proving, then analyzes the impact of the long-tailed tactic distribution on proof synthesis, motivating our work.

2.1 Search-based Neural Theorem Proving

Neural Theorem Proving simulates the interaction between human proof engineers and proof assistants, and employs a meta-heuristic search procedure to construct complete proof scripts incrementally. A proof script consists of a sequence of proof steps, each corresponding to a tactic applied to the current proof state. During the search process, the system starts with an empty script and predicts a set of candidate tactics (e.g., $n = 20$). These candidates are then verified one by one by interacting with the proof assistants (e.g., Coq) to check whether they can be successfully applied to the current state. If a candidate tactic fails to compile or does not produce a new valid state, the search continues with the next candidate; if it succeeds, the tactic is appended to the script and the process iterates until all subgoals are resolved. Once the proof state contains no remaining goals, the prover finalizes the script with the `Qed` command. If the search exceeds a time or step limit, it is considered a failure.

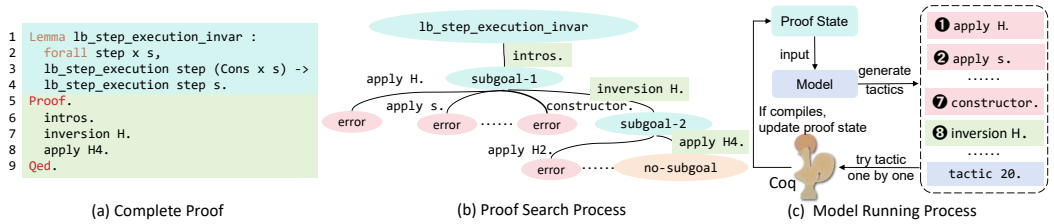
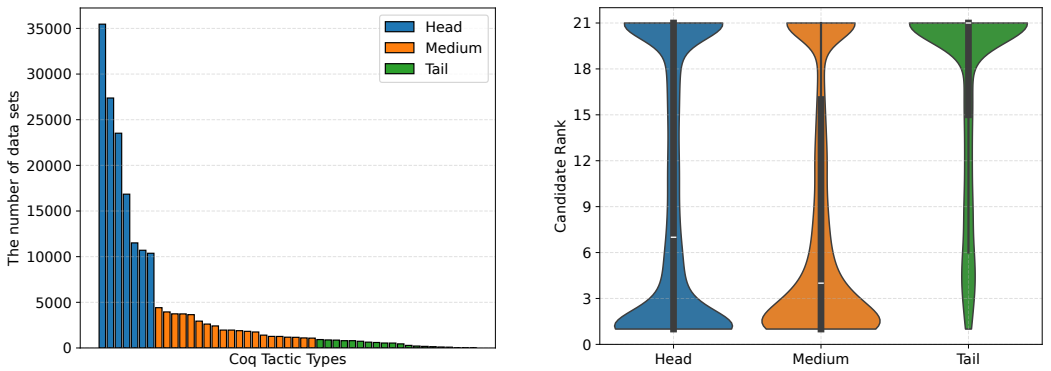


Fig. 1. Neural Theorem Proving Workflow on the Verdi Lemma Example

Figure 1 illustrates this process with a concrete example from the Verdi framework for formally verified distributed systems, namely the lemma `lb_step_execution_invar`. This lemma captures an invariance property of the label-based semantics, specifically that if a step holds on an execution trace extended with a head element, it also holds on the trace without that element. As shown in Figure 1(a), lines 1–4 define the lemma, while lines 6–8 show the complete proof script, consisting of three tactics: `intros`, `inversion H`, and `apply H4`. Figures 1(b) and 1(c) present the proof search performed by the neural theorem proving in constructing this proof script. Initially, the proof state corresponds to the lemma itself. Given this state, the model generates 20 candidate tactics, and the first candidate, `intros`, is executed by the Coq proof assistant, producing a new subgoal and leading to the second proof step. In the second step, the model again generates 20 candidates. Several initial attempts (e.g., `apply H`, `apply s`) are invalid and result in an error by the Coq proof assistant. The eighth candidate, `inversion H`, is successfully executed and produces a new goal. In the third proof step, the tactic `apply H4` eliminates the final subgoal, after which the Coq proof assistant confirms completion and terminates the script with `Qed`. This example demonstrates the collaborative process between the neural theorem proving and the Coq proof assistant. The model generates candidate tactics, which are subsequently validated by the Coq environment. Through iterative search, they eventually converge to a correct and complete proof script.

2.2 Long-tailed Problem in Neural Theorem Proving

To investigate whether the long-tailed distribution of tactic types in the dataset affects tactic generation, we conduct an empirical analysis based on the CoqGym dataset [65] and the representative neural theorem prover ASTactic [65] trained on it. Following the standard partitioning used in long-tailed scenario [18, 61, 73], we divide the tactic types in the training and validation sets of CoqGym into three categories according to their frequencies: head (more than 5000 samples per type), medium (1000–5000 samples), and tail (fewer than 1000 samples). Under this partition, the head class contains 7 tactics, while the medium and tail classes each contain 20 tactics. As shown in Figure 2a, a few head-class tactics (e.g., apply, intro, rewrite) dominate the dataset, typically serving fundamental roles such as introducing assumptions, applying known results, or simplifying goals. In contrast, tail-class tactics (e.g., inversion_clear, symmetry, lia) occur much less frequently but play indispensable roles in inductive reasoning, equality manipulation, and arithmetic proofs.



(a) Long-tailed distribution of tactic types

(b) Reference tactic positions across tactic categories

Fig. 2. Distribution and generation performance across long-tailed tactic categories

Based on this partition, we evaluate ASTactic on proof steps from the CoqGym test set. Each proof step is paired with a human-written reference tactic, and for each step, we generate 20 candidate tactics and record the rank position of the reference tactic. Figure 2b reports the ranking results across different categories. We observe that, relative to head and medium classes, reference tactics belonging to the tail class are more likely to occur in later positions within the candidate list, indicating that the model exhibits difficulty in generating low-frequency tactics. Moreover, even within the head and medium classes, a considerable proportion of reference tactics are assigned relatively low ranks, indicating that generation is affected by ranking inconsistencies across categories. These findings demonstrate that the long-tailed distribution not only reflects inherent frequency disparities in tactic usage but also biases the model toward high-frequency tactics while neglecting low-frequency yet essential ones, thereby compromising proof completeness and diversity. This observation underscores the necessity of developing effective tactic reranking methods, which directly motivates our subsequent retrieval–generation fusion approach.

3 Approach

In this section, we present the proposed ProofFusion, an adaptive retrieval-augmented reasoning framework that improves the performance of neural theorem proving models without retraining. As illustrated in Figure 3, ProofFusion consists of two key phases: proof semantic-aware retriever

training and adaptive retrieval-augmented reasoning. The retriever is built on a proof semantic alignment dataset and fine-tuned from CodeBERT [10] using contrastive learning [26]. It performs a proof semantic search over the knowledge base to identify historical proof goals and their corresponding tactic that exhibit similarity, producing a traceable reference set. In the adaptive retrieval-augmented reasoning phase, the initial theorem proof state is processed by a capability-adaptive retrieval classifier, which determines whether retrieval should be invoked. When retrieval is employed, a dual-track reranking fusion mechanism integrates the neural model's original outputs with the retrieved reference tactics; otherwise, the model-generated candidate tactics are utilized directly. The proof assistant sequentially applies the candidate tactics to advance the proof and generate new subgoals, and this iterative procedure continues until all subgoals are resolved.

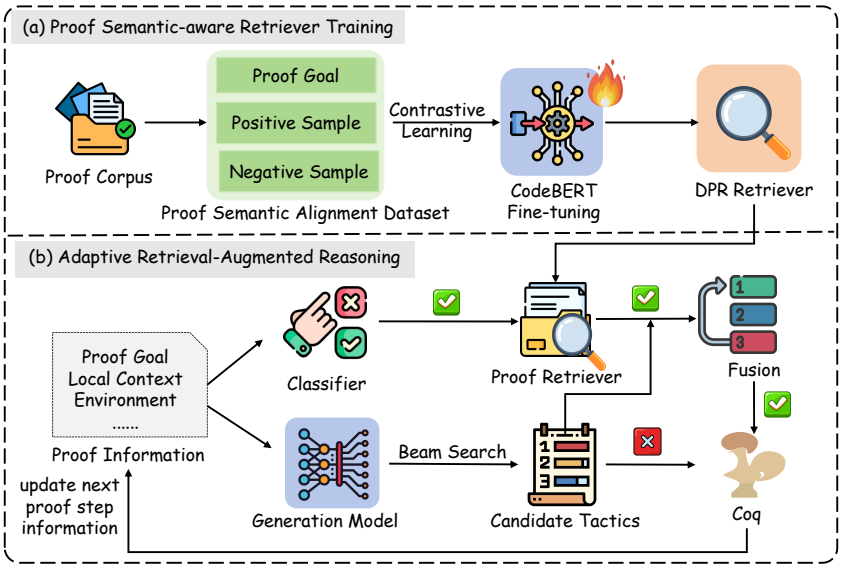


Fig. 3. Overview of the adaptive retrieval-augmented reasoning framework

3.1 Proof Semantics-aware Retriever

The proof semantics-aware retriever in ProofFusion addresses the limitations of existing retrieval methods [57], which primarily rely on shallow semantic similarity [11]. Shallow similarity alone is often insufficient, as two goals may appear similar but depend on entirely different tactics, limiting the effectiveness of retrieval for tactic guidance (Challenge 1). To more accurately capture transferable tactics, we leverage the syntactic information of tactics in historical proof data to construct a Proof Semantic Alignment Dataset. Inspired by formal proofs, the syntactic form of a tactic often reflects the logical structure and proof semantic intent [1] of the corresponding proof goal. By normalizing and modeling tactic syntax, the retriever can identify goals with consistent syntax, which are more likely to share reasoning patterns and enable tactic transfer. To this end, we adopt syntactic consistency as the alignment criterion, rather than depending solely on tactic names, which often under-specify semantic intent. For example, `induction n` and `induction m using IHm` are both categorized as `induction` tactics, yet their syntactic forms differ, which results in distinct unfolding behaviors and consequently different semantic structures. Alignment based solely on tactic names would thus introduce semantic noise. The proof semantics-aware retriever consists of two main stages: *Proof Semantic Alignment Dataset Construction* and *Retriever Training Phase*. These stages are described in detail below.

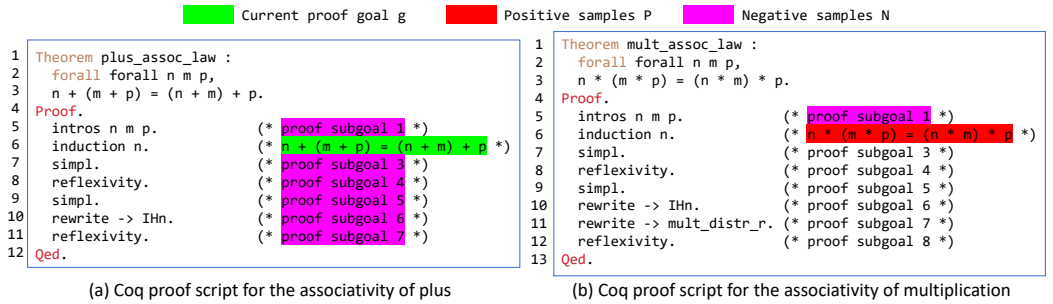
3.1.1 Proof Semantic Alignment Dataset Construction. In this stage, we construct the proof semantic alignment dataset, which subsequently serves to the training data to train a DPR model [23]. The dataset is derived from the training and validation splits of CoqGym [65], which provides complete proof traces, each consisting of a proof goal, its associated tactic, and the corresponding theorem name. This corpus therefore forms a reliable and large-scale source of historical proof data for training retrieval models. For DPR, each proof goal is regarded as the fundamental unit of retrieval, and the syntax of its tactic is normalized to mitigate noise introduced by syntactic variations. To satisfy the input specifications of the DPR model, both positive and negative samples are constructed for each proof goal. Positive samples consist of proof goals with syntactically consistent tactics as relevant examples, while negative samples consist of goals with inconsistent tactics to guide the model in distinguishing fine-grained semantic differences between proof goals. The process of constructing the proof semantic alignment dataset is summarized in Algorithm 1. Each dataset entry is represented as a triplet (g, P, N), where g denotes a proof goal, P its positive sample, and N the corresponding set of negative samples. The construction proceeds through two principal procedures: positive sample construction and negative sample construction.

Algorithm 1: Construction Process of the Proof Semantic Alignment Dataset

Input: Set of proof entries $\mathcal{E}$, similarity function $\text{sim}(\cdot,\cdot)$, number of negative samples N_{neg}
Output: DPR dataset $\mathcal{D}$

Initialize dataset $\mathcal{D} \leftarrow \emptyset$
for each proof goal $g \in \mathcal{G}$ **do**
  Let t_g be the tactic used for g
  `// Positive sample construct`
  $G_{\text{syn}} = \{g' \in \mathcal{G} \mid \text{syntax}(t_{g'}) = \text{syntax}(t_g)\}$ `// Semantically consistent proof goals`
  Compute similarity for theorem name: `// Find the semantically closest proof goal`
  $sim(g') \leftarrow \text{sim}(\text{theorem\_name}(g), \text{theorem\_name}(g'))$ for all $g' \in G_{\text{syn}}$
  $P \leftarrow$ proof goal(s) in G_{syn} with highest $sim(g')$ `// Positive sample`
  if $P = \emptyset$ **then**
    continue `// Skip if no positive sample found`
  end if
  `// Negative sample construct`
  Initialize negative sample set $N \leftarrow \emptyset$
  $F \leftarrow$ proof goals in the same theorem as g with different tactic syntax
  Truncate F to N_{neg} if $|F| > N_{neg}$
  $N \leftarrow N \cup F$
  `// Initial negative samples are insufficient`
  if $|N| < N_{neg}$ **then**
    $S \leftarrow$ all proof goals $g' \in \mathcal{G} \setminus \{g\}$ with different tactic syntax
    Sort S by descending $\text{sim}(\text{theorem\_name}(g), \text{theorem\_name}(g'))$
    for proof goals in S until $|N| \geq N_{neg}$ **do**
      Add to N
    end for
  end if
  Add (g, P, N) to $\mathcal{D}$
end for
return $\mathcal{D}$

Positive Sample Construction. For positive samples, we select proof goals that have the same tactic syntactic form and are associated with theorems that are semantically related. Tactics with consistent syntactic forms generally correspond to similar logical operations, thereby making their associated proof goals more semantically related. For each proof goal g, we first identify its tactic t_g and collect all proof goals whose tactics share the same syntactic form, forming a candidate set G_{syn} (line 4). However, tactic consistency alone is insufficient to identify the most relevant positive sample, since proof goals from different theorems may still differ significantly in semantics. To address this (lines 5–7), we introduce a *theorem name similarity function* $\text{sim}(\cdot,\cdot)$ to measure the semantic relevance between the theorem of g and those of candidate goals. Before computing similarity, theorem names are preprocessed by replacing underscores (_) with spaces, splitting CamelCase and letter-number boundaries, and converting the apostrophe (') to the word 'prime". This converts programmatic theorem identifiers into natural-language-like sequences suitable for semantic embedding. We then encode the processed theorem names using the sentence embedding model `all-MiniLM-L6-v2` [60], which is lightweight yet effective for capturing semantic similarity between short textual sequences. The similarity score between two theorems is computed as the cosine similarity between their embeddings. The goal in G_{syn} whose theorem name is most similar to that of g is selected as the unique positive sample P. As illustrated in Figure 4, consider the second proof step of the addition associativity theorem, whose goal $n + (m + p) = (n + m) + p$ serves as the proof goal g for constructing positive and negative samples. Its most semantically similar theorem is the multiplication associativity theorem. Both g and the second tactic of the multiplication associativity theorem use the same syntactic form of tactic `induction n`, and thus the positive sample for g is $n * (m * p) = (n * m) * p$. If no such sample exists (lines 8–10), the proof goal is excluded from dataset construction.



(a) Coq proof script for the associativity of plus

(b) Coq proof script for the associativity of multiplication

Fig. 4. Dataset construction example for proof semantic alignment

Negative Sample Construction. Negative samples provide contrastive information that is *semantically related but not fully consistent*, with the core objective of constructing *hard negative samples*. Compared with trivially unrelated negatives, hard negatives pose greater challenges for the retriever and thus provide stronger contrastive learning signals [23, 46, 64]. To this end, proof goals from the same theorem as the target goal g but with differing tactic syntax are chosen as the primary negative samples (lines 11–14). We prioritize such goals because proof states within the same theorem are highly correlated, and those employing different syntactic tactics are particularly difficult to distinguish, thereby serving as representative hard negatives. If their number exceeds N_{neg}, the set is truncated to N_{neg}; if insufficient, the selection is expanded to the global dataset (lines 15–21), where candidates are restricted to goals with different tactic syntax and ranked by theorem name similarity to g until N_{neg} samples are obtained. As illustrated in Figure 4, for a target goal g, we first select negative samples from other proof goals within the same theorem. If this

set is insufficient, we further supplement negative samples by choosing proof goals with differing tactic syntax from theorems whose names are most similar to that of g.

3.1.2 Retriever Training Phase. Our objective is to develop an efficient DPR for identifying semantically related proof goals. Unlike standard DPR framework [49], which uses separate encoders for heterogeneous inputs, our task involves homogeneous proof goals. This setting allows us to employ a single CodeBERT [10] model to encode both the query (current proof goal) and context (candidate historical proof goals). Such a design enables parameter sharing, motivates the adoption of a Siamese network [35], and ensures representational consistency while reducing redundancy. To learn DPR representations from queries with one positive and multiple negative samples, we adopt `MultipleNegativesRankingLoss`. This loss encourages each query to be closer to its positive than to the negatives, effectively capturing fine-grained semantic distinctions without explicit negative sampling.

For training, we follow the configuration used in DPR [23, 39]. Specifically, each proof goal serving as a query is paired with one semantically related positive sample and up to seven negative samples. This design exposes the model to both similar and dissimilar instances while mitigating excessive noise, thereby balancing efficiency with the strength of the training signal. The dataset is partitioned in an 8:2 ratio for training and validation. Model evaluation is performed using the `InformationRetrievalEvaluator` module from the `sentence-transformers` framework, with cosine similarity-based MRR@10 employed to assess the quality of the top-10 retrieved results. This metric directly reflects the retrieval objective by emphasizing the rank position of the first correct answer in the returned list. To ensure stable and robust performance, early stopping is applied. Training is terminated if the validation metric does not improve over five consecutive evaluations, and the checkpoint achieving the best validation performance is preserved.

3.2 Adaptive Retrieval-augmented Reasoning

Adaptive retrieval-augmented reasoning presents two central challenges: how to integrate retrieval outputs with generative results in a manner that is both effective and explainable (Challenge 2), and how to determine whether retrieval should be invoked adaptively (Challenge 3). To address these challenges, we introduce a dual-track reranking fusion mechanism together with a capability-aware retrieval classifier tailored for theorem proving. The subsequent sections provide a detailed description of these two components.

3.2.1 Dual-track Reranking Fusion Mechanism. Most existing retrieval-augmented generation methods adopt a traditional query-based fusion paradigm [62, 72], in which retrieved documents are concatenated with the original query and subsequently processed by the generation model. Although straightforward, this strategy suffers from input length constraints and offers limited explainability. To overcome these limitations, we propose an output-probability-based fusion mechanism [24, 25]. In this approach, the information retrieved is incorporated in the model output layer, rendering the provenance of each prediction transparent. Notably, this design preserves compatibility with existing generation models and does not require architectural modifications or retraining. The following section elaborates on the conceptual motivation and methodological details of the proposed fusion mechanism.

High-level idea. To enhance the explainability of retrieval-augmented tactic generation, we embed the neural theorem prover as a predicate within the logical system [8], thereby grounding the outputs of the prover in formal logical explanation. Formally, as shown in Equation 1, tactic generation integrates the outputs of the neural prover with the retrieved tactics. As shown in Figure 5, the tactic produced by the neural prover is conjunctively combined with the tactic returned by the retriever to form the final tactic prediction. This design allows each generated tactic to be

traced back to semantically related historical proof data retrieved from the database, including the corresponding proof goal, the theorem name associated with the goal, as well as the source file and project name. These retrieved contextual signals provide a principled foundation for explainable tactic selection and help users better understand the historical proof data underlying the prediction. For each proof step, we match the tactic generated for the current proof goal with the tactics returned by the retrieval module. Specifically, the generative model produces candidate tactics based on the current goal, while the retriever provides semantically similar historical proof goals together with the tactics used in their original proofs. If a retrieved tactic matches the generated one in syntactic form or shares the same tactic type, we regard them as aligned and trace the generated tactic back to the corresponding historical proof goal. This alignment mechanism also motivates the design of our proof semantic-aware retriever, which must be capable of identifying tactics associated with semantically related proof goals to support explainable tactic generation.

$$R[T]\ (Gen(g,t) \wedge t \in T \ \rightarrow\ F(t \mid g)) \tag{1}$$

where $R[T](\cdot)$ denotes the set of tactics T retrieved by the retriever given the current proof goal g. $Gen(g,t)$ denotes the candidate tactic t generated by the model for the proof goal g. $t \in T$ indicates that t belongs to the retrieved tactic set T. Finally, $F(t \mid g)$ represents the fused tactic t obtained for the proof goal g.

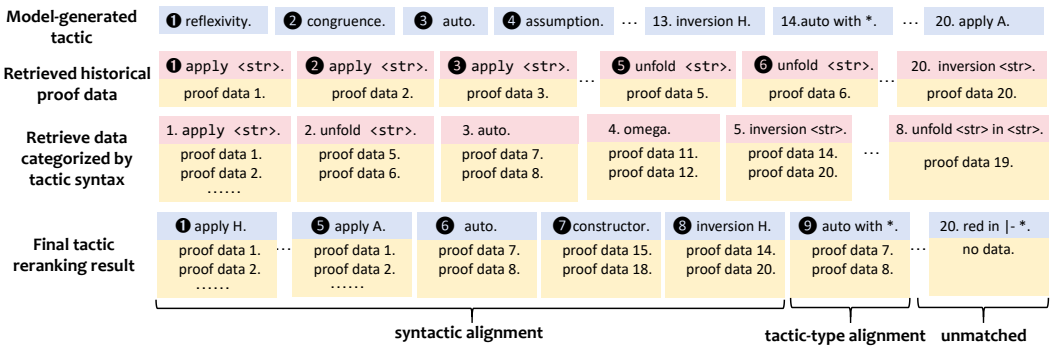


Fig. 5. An example of reranking fusion for tactic prediction in a proof step

Details of the fusion mechanism. We integrate retrieved tactics with generated candidates through a hierarchical dual-track reranking strategy. As shown in Figure 5, for each proof goal, we retrieve 20 semantically related historical goals and their associated tactics, keeping the number of retrieved tactics consistent with the generated ones. The reranking proceeds in three stages: (i) generated tactics that share the same syntactic form as retrieved tactics are aligned and ranked by retrieval similarity, (ii) the remaining unmatched tactics are then aligned by tactic type and ranked by retrieval similarity, and (iii) tactics that cannot be matched in either stage retain their original generation probabilities to preserve model confidence. Syntactic alignment provides the strongest semantic relevance to the proof goal and is therefore prioritized in the first stage. However, relying on syntax alone risks discarding useful tactics, so tactic-type alignment is introduced to extend the coverage of relevant candidates while maintaining explainability. This layered strategy strikes a balance between precise semantic alignment, functional relevance, and model confidence. To assess the contribution of each stage, we further evaluate syntax-only and type-only variants in Section 4.4.

3.2.2 Capability-adaptive Retrieval Classifier. Since retrieval may introduce noise [9], it is essential to adopt an adaptive mechanism capable of determining when retrieval augmentation is necessary.

Existing adaptive retrieval-augmented methods typically rely on a complexity classifier [22] to automatically decide whether to invoke retrieval. However, constructing such classifiers usually requires historical complexity annotations for each proof goal, which are often unavailable in practice. To address this limitation, we instead analyze the proof search histories of four neural theorem proving tools [13, 52, 65] used in this work. Our analysis reveals that for simple theorems that the tools can automatically prove, the number of tactic attempts at each proof step is very limited. In contrast, for complex theorems that the tools cannot directly prove, each proof step involves attempting a large number of tactics. This consistent observation across different tools suggests that the finding is generalizable. Based on this observation, we conclude that for simple theorems, the tools themselves provide sufficient reasoning capability, and retrieval is unnecessary, as language model generation alone suffices. For more challenging theorems, however, retrieval augmentation is needed at every proof step.

Motivated by this observation, we automatically label the difficulty of theorems in the validation set by checking how the tools perform on them. In practice, each theorem is automatically labeled as either "easy" or "hard" based on whether the tools can complete a proof. This approach avoids reliance on manual annotation while accurately reflecting the reasoning capability of the tools on different theorems. Using these labeled data, we train a lightweight capability-adaptive classifier to dynamically decide whether to apply retrieval during proof search. After experimenting with several machine learning and deep learning models commonly used for text classification [35], we select TextCNN [27] as the final classifier due to its superior performance. A detailed comparison of these models is provided in Section 4.5. Once trained, the classifier is applied at the beginning of proof search to predict the difficulty of each theorem. For simple theorems, retrieval is disabled throughout the search, whereas for complex theorems, retrieval is applied at every proof step.

4 Evaluation

This experiment aims to evaluate the effectiveness of ProofFusion in enhancing existing neural theorem provers for proof synthesis. Specifically, we seek to address the following five research questions:

- **RQ1:** How effective are existing neural theorem proving methods enhanced via ProofFusion on proof synthesis?
- **RQ2:** How does the retriever of ProofFusion compare with other methods?
- **RQ3:** How effective are reranking methods in retrieval–generation fusion of ProofFusion?
- **RQ4:** How effective is the adaptive retrieval module of ProofFusion?
- **RQ5:** To what extent can ProofFusion explain synthesized proofs?

4.1 Experimental Setup

4.1.1 Benchmark. We utilize CoqGym [65], a state-of-the-art benchmark for evaluating neural theorem proving systems. CoqGym contains 70,856 theorems from 123 open-source Coq projects and provides a structured learning environment integrated with the Coq proof assistant, enabling rigorous evaluation under realistic conditions. Consistent with prior studies, our baselines are trained and evaluated on the designated training and test sets of CoqGym. The training and validation sets comprise 96 projects with 57,719 theorems and manually authored proof scripts, while the test set consists of 13,137 theorems from the remaining 27 projects. Previous attempts to reproduce neural proving performance encountered errors in the internal handling of proof scripts by Coq, particularly affecting the coq-library-undecidability project. To ensure rigor and reproducibility, this project is excluded. Consequently, our assessment is conducted on 10,782 theorems from 26 projects for which proof scripts can be reliably executed.

4.1.2 Baselines. We consider four state-of-the-art neural theorem provers, *ASTactic* [65], *Tac* [13], *Tok* [13], and *Passport* [52]. Specifically, *ASTactic* encodes proof states with a TreeLSTM and generates syntactically valid tactics. *Tac* and *Tok* integrate partial proof scripts with proof states. In *Tac*, the encoder parses only common Coq tactic name tokens, whereas *Tok* parses the entire token sequence except for punctuation. *Passport* extends existing Coq tactic prediction models by introducing three additional identifier encoding mechanisms. Notably, this study investigates whether ProofFusion can enhance the proof capabilities and explainability of neural theorem provers without requiring retraining. Therefore, evaluating whether it surpasses the current state of the art is not our primary concern. We believe that if ProofFusion can improve the performance of these selected provers, it is likely to benefit other neural theorem provers as well.

4.1.3 Evaluation Metrics. Following the previous evaluation [5, 12, 13, 52], we use **Total Added Value** and **Unique Added Value** to assess how much ProofFusion improves the performance of the neural proof synthesis tools of the baseline. Their definitions are as follows:

Total Added Value. This metric measures the overall improvement achieved by ProofFusion. It is defined as the increase in the number of theorems proved when ProofFusion is added to the baseline tool, divided by the number of theorems proved by the baseline tool. The result shows the percentage improvement in theorem proving capability.

Unique Added Value. This metric measures the specific advantage brought by ProofFusion. It is defined as the number of theorems that can only be proved with ProofFusion, divided by the number of theorems proved by the baseline tool. The result shows the percentage of entirely new theorems contributed by ProofFusion.

4.1.4 Implementation Details. We implement retrieval using Pyserini [38], a Python interface to Apache Lucene, allowing efficient indexing and searching of textual data. For vector-based DPR retrieval, we use Facebook AI Similarity Search (FAISS) to construct a vector index. Since FAISS stores only vectors, we maintain a Lucene index aligned with the same IDs to ensure proof traceability and enhance explainability. Neural theorem provers retain their original architectures and default hyperparameters. The proof search follows prior work, with a 10-minute limit per theorem, up to 20 candidate tactics per step, a maximum of 300 tactic attempts per theorem, and solutions limited to 50 tactics. Evaluation within the Coq proof environment is computationally intensive, but proofs are environment-independent, allowing parallel execution. For CoqGym experiments, theorems are partitioned by project files and evaluated on an HPC server managed by Slurm, equipped with dual AMD 7285H 32-core processors, 256 GB DDR4 memory, and a 2.5 GHz base clock. The experiments required roughly 10,000 CPU hours, costing approximately $140.

4.2 RQ1: How effective are existing neural theorem proving methods enhanced via ProofFusion on proof synthesis?

Table 1 presents the effect of integrating ProofFusion into four neural theorem provers. For each tool, incorporating the ProofFusion framework substantially improves theorem proving capability. Specifically, in terms of Total Added Value, ProofFusion increases the number of theorems proved by ASTactic by 5.4%, adding 71 additional theorems; Tac by 9.65%, adding 108 theorems; Tok by 6.50%, adding 76 theorems; and Passport by 6.00%, adding 72 theorems. On average, ProofFusion improves proof success rates by 6.89%, corresponding to 82 additional theorems. These results indicate that ProofFusion systematically enhances neural theorem proving without requiring retraining of the underlying models.

Following the evaluation methodology of previous neural theorem proving tools, we further investigate whether ProofFusion can enable provers to solve new theorems. In terms of Unique Added Value, the framework demonstrates a particularly notable impact. Specifically, ProofFusion

Table 1. Performance Improvement of Neural Theorem Provers with ProofFusion

	Default	ProofFusion	Total Added Value	Unique Added Value
ASTactic	1,316	1,387	71 (5.40%)	161 (12.23%)
Tac	1,119	1,227	108 (9.65%)	247 (22.07%)
Tok	1,169	1,245	76 (6.50%)	233 (19.93%)
Passport	1,200	1,272	72 (6.00%)	189 (15.75%)
Average	-	-	82 (6.89%)	208 (17.50%)

allows ASTactic to prove 161 theorems previously unprovable, yielding 12.23% unique added value; Tac proves 247 new theorems, corresponding to 22.07%; Tok proves 233 new theorems, corresponding to 19.93%; and Passport proves 189 new theorems, corresponding to 15.75%. Overall, ProofFusion enables an average of 208 additional new theorems across all provers, with an average unique added value of 17.50%. These findings highlight that ProofFusion not only enhances existing capabilities but also enables provers to discover entirely new theorems, confirming its effectiveness as a complementary mechanism for improving proof synthesis.

In addition, since ProofFusion is built as an enhancement to existing neural theorem provers, while it can prove some theorems that previous tools cannot, there still exist theorems that can be proved by other tools but not by ProofFusion. To understand the reasons behind such cases, we analyze these theorems and identify two main factors. First, these theorems generally require deeper proof searches, with baseline models averaging about 12 tactic steps compared to 7 for other theorems. Consequently, such proofs are more sensitive to small deviations during the search process. Second, neural theorem provers often generate many invalid tactic parameters. In deep search scenarios, this can cause a large number of attempts to be wasted on invalid tactics, increasing the likelihood of reaching the preset maximum number of steps.

Result of RQ1: Integrating ProofFusion into existing neural theorem provers consistently improves their performance, increasing both the total number of theorems proved and the number of entirely new theorems discovered. On average, it enhances proof success rates by 6.89% and enables 17.50% of theorems that were previously unprovable.

4.3 RQ2: How does the retriever of ProofFusion compare with other retrieval methods?

To comprehensively evaluate the effectiveness of the proposed retriever, we compared it with three baseline methods: the traditional BM25 retriever [50] based on sparse representations, a dense retriever built on CodeBERT, and hybrid retrieval approach. In the experiments, manually written tactics from the test set were used as reference answers. For each query, the top 20 retrieved candidates were examined, and the position of any candidate matching the reference tactic in the ranking was recorded. Since our goal is to align the retrieval results as closely as possible with the human-written tactics, the retriever is expected not only to recall the correct results but also to rank them among the top positions. Accordingly, we adopted standard evaluation metrics from information retrieval, namely Precision@k ($P@k$), Mean Average Precision (MAP), and Mean Reciprocal Rank (MRR), to measure retrieval performance. Precision@k quantifies the proportion of relevant results within the top-k candidates, MAP reflects the average ranking position of all relevant results across the retrieval list, and MRR captures the rank of the first correct result.

Table 2 presents the performance comparison of the proposed retriever ProofFusion with three baseline methods. Overall, ProofFusion achieves the highest scores across all evaluation metrics, indicating its superior ability to retrieve results that closely align with human-written

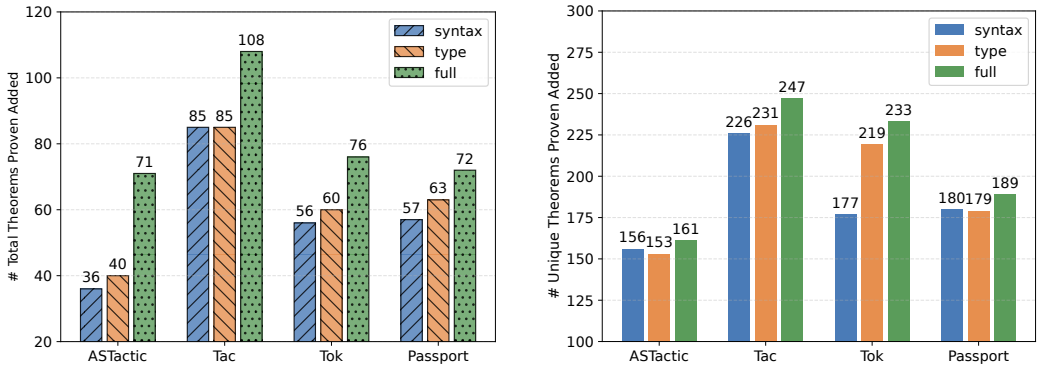
Table 2. Retrieval Performance of ProofFusion and Other Retrieval Methods

	P@1	P@5	P@10	P@20	MAP	MRR
BM25	0.1371	0.1365	0.1320	0.1255	0.1893	0.2151
CodeBERT	0.0166	0.0254	0.0257	0.0315	0.0557	0.0583
BM25_CodeBERT	0.1371	0.0970	0.0810	0.0797	0.1489	0.2002
BM25_ProofFusion	0.1783	0.1746	0.1744	0.1709	0.2324	0.2817
ProofFusion	**0.2256**	**0.2106**	**0.2095**	**0.2069**	**0.2679**	**0.3079**

tactics. Specifically, ProofFusion attains *P*@1 of 0.2256, substantially outperforming BM25 with 0.1371, CodeBERT with 0.0166, and the BM25_CodeBERT hybrid with 0.1371, suggesting that the correct tactic is more likely to be ranked first. Consistent improvements are also observed for *P*@5, *P*@10, and *P*@20, demonstrating that the retriever not only recalls relevant tactics but also ranks them favorably. *MAP* and *MRR* reach 0.2679 and 0.3079, respectively, further confirming the advantage of ProofFusion in overall ranking quality and the position of the first correct result. Notably, the hybrid retrieval combining BM25 with ProofFusion performs worse than ProofFusion alone, likely because BM25 relies on keyword matching and may introduce low-relevance candidates that undermine the semantic retrieval capability of ProofFusion. Taken together, these results demonstrate that ProofFusion yields substantial improvements in both the alignment with human-authored tactics and the accuracy and quality of the retrieval rankings.

Result of RQ2: ProofFusion outperforms all other retrieval methods across all evaluation metrics, achieving more accurate retrieval and higher-ranking of results that align with human-written tactics. The results demonstrate its significant advantage in both retrieval precision and overall ranking quality.

4.4 RQ3: How effective are reranking methods in the retrieval–generation fusion of ProofFusion?



(a) Total theorems added by reranking methods (b) Unique theorems added by reranking methods

Fig. 6. Comparison of improvements in proving performance across different reranking fusion mechanisms

To evaluate the effectiveness of our reranking fusion mechanism, we compared it with two variants that rely solely on syntax reranking and type reranking. Figure 6 presents the results of the three reranking fusion mechanisms on improving the proving capability of neural theorem

Table 3. Effect of the adaptive retrieval module on ProofFusion across four neural theorem provers

Model	Default	Total Added Value		Unique Added Value	
		w/o	w	w/o	w
ASTactic	1316	50 (3.80%)	71 (5.40%)	171 (12.99%)	161 (12.23%)
Tac	1119	102 (9.12%)	108 (9.65%)	247 (22.07%)	247 (22.07%)
Tok	1169	66 (5.65%)	76 (6.50%)	219 (18.73%)	233 (19.93%)
Passport	1200	65 (5.42%)	72 (6.00%)	180 (15.00%)	189 (15.75%)
Average	-	71 (6.00%)	82 (6.89%)	204 (17.20%)	208 (17.50%)

provers. The syntax-only variant reranks candidates by prioritizing tactic syntax matches, the type-only variant reranks by prioritizing tactic type matches, and the full method corresponds to our proposed reranking fusion mechanism that combines both. Overall, the Dual-track Reranking Fusion Mechanism consistently outperforms the other two approaches across all neural theorem provers. In Figure 6a, it achieves the highest number of additional proved theorems, and in Figure 6b, it also yields the largest increase in previously unsolved theorems. For example, in ASTactic, our method proves 71 additional theorems and 161 previously unsolved ones, compared to only 36 and 156 for the syntax-only variant and 40 and 153 for the type-only variant. Similar trends are observed in Tac, Tok, and Passport, where the dual-track reranking fusion mechanism not only increases the overall number of proved theorems but also expands the set of previously unsolved theorems. These results demonstrate that integrating syntax and type retrieval signals through a layered fusion strategy is substantially more effective than relying on a single source of information.

Result of RQ3: The dual-track reranking fusion mechanism consistently outperforms variants relying solely on syntax or type information across all neural theorem provers, yielding substantial improvements in both the total number of proved theorems and the set of previously unsolved theorems through the integrated use of syntax and type signals.

4.5 RQ4: How effective is the adaptive retrieval module of ProofFusion?

To evaluate the effectiveness of the adaptive retrieval module, we conducted an ablation study comparing ProofFusion with (w) and without (w/o) this module across four neural theorem provers. Table 3 summarizes the results under both settings. It can be observed that incorporating the adaptive retrieval module consistently enhances the proving capability of ProofFusion. In terms of Total Added Value, all four provers show improvements: ASTactic increases from 50 (3.80%) to 71 (5.40%), Tac from 102 (9.12%) to 108 (9.65%), Tok from 66 (5.65%) to 76 (6.50%), and Passport from 65 (5.42%) to 72 (6.00%). The overall average also increases from 71 (6.00%) to 82 (6.89%), indicating a consistent and robust improvement. Regarding Unique Added Value, although ASTactic slightly decreases from 171 (12.99%) to 161 (12.23%), the other models remain unchanged or improve: Tac remains at 247 (22.07%), Tok increases from 219 (18.73%) to 233 (19.93%), and Passport increases from 180 (15.00%) to 189 (15.75%). Consequently, the overall average increases from 204 (17.20%) to 208 (17.50%). Overall, the adaptive retrieval module delivers stable gains in Total Added Value and extends the capability of previously unproven theorems for most models, demonstrating its generality and robustness in the fusion of retrieval and generation.

In addition, introducing an extra retrieval module during proof search may increase the complexity of the system, so we also evaluated its impact on runtime. Figure 7 compares the average proof time per theorem with and without the retrieval module. Surprisingly, the retrieval module not only avoids increasing the search time but also brings slight improvements in most provers.

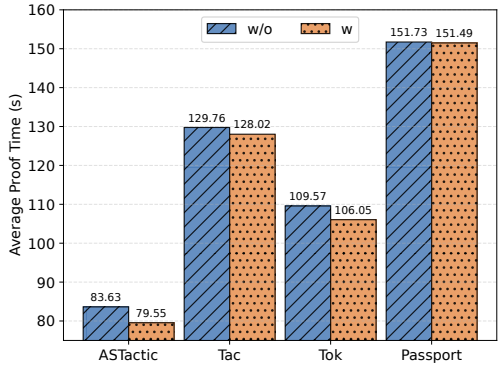


Fig. 7. Impact of the adaptive retrieval module on average proof time per theorem

Specifically, ASTactic decreases from 83.63 seconds to 79.55 seconds, Tac from 129.76 seconds to 128.02 seconds, Tok from 109.57 seconds to 106.05 seconds, while Passport remains almost unchanged at 151.73 seconds compared to 151.49 seconds. Overall, the adaptive retrieval module improves search efficiency without adding runtime overhead, further demonstrating its practicality and effectiveness in real proof tasks. We attribute this phenomenon to the capacity of the module to generate more targeted candidate tactics, thereby reducing the number of search attempts required to identify a feasible proof path and consequently shortening the overall search time. Put differently, the adaptive retrieval module not only enhances the proof success rate but also improves the efficiency of the search process.

Table 4. Performance comparison of different classifiers for tool capability-adaptive

Model	Accuracy	Precision	Recall	F1-score
KNN	86.78%	69.70%	65.44%	67.14%
LR	88.10%	81.13%	55.49%	56.76%
NB	87.75%	82.42%	53.41%	53.13%
RF	90.02%	82.42%	66.58%	70.94%
SVM	88.64%	**84.18%**	57.67%	60.13%
BiLSTM	89.21%	70.23%	65.84%	67.08%
BiLSTM_Att	89.40%	80.11%	66.92%	70.47%
Codebert	88.91%	76.72%	68.00%	70.67%
FastText	87.35%	52.59%	51.38%	49.14%
TextCNN	**90.41%**	82.07%	**70.06%**	**73.87%**

We further compared the performance of classifiers used in the adaptive retrieval module to evaluate their effectiveness in theorem difficulty classification tasks. Experiments were conducted on a range of common machine learning and deep learning text classifiers, and Table 4 presents their average performance across four tools. We found that TextCNN achieved the best overall performance. Specifically, TextCNN attained an Accuracy of 90.41%, a Precision of 82.07%, a Recall of 70.06%, and an F1-score of 73.87%, outperforming all other models across these metrics. In contrast, traditional machine learning methods such as KNN and logistic regression exhibited less balanced performance between precision and recall, while deep learning methods including BiLSTM, BiLSTM_Att, and CodeBERT showed strong performance on some metrics but still had lower overall F1-scores compared to TextCNN. SVM achieved precision close to that of TextCNN,

but its Recall and F1-score lagged behind, indicating a less comprehensive capability to identify tool effectiveness. These results demonstrate that TextCNN can consistently capture capability features across different neural theorem provers, as its convolutional filters are particularly effective in extracting local syntactic and semantic patterns from proof goals, which are highly indicative of theorem difficulty. This enables a more accurate prediction of the potential contribution of each tool during the proof search of the theorem.

Result of RQ4: The adaptive retrieval module consistently improves ProofFusion's proving capability across all models while slightly reducing average proof time, demonstrating that it enhances both proof success and search efficiency without adding runtime overhead.

4.6 RQ5: To what extent can ProofFusion explain synthesized proofs?

To evaluate the impact of ProofFusion on the explainability of neural theorem provers, we conducted an analysis on theorems successfully proved using the retrieval module. Specifically, we examined the tactics applied at each proof step. A proof step was considered explainable if the tactic could be traced back to a retrieved proof set. Figure 8 shows the proportion of explainable proof goals across the four neural provers. Concretely, ASTactic had 5,976 explainable goals out of 7,169 total goals, corresponding to 83.36%, Tac had 6,764 explainable goals out of 8,209 total goals (82.40%), Tok had 6,171 explainable goals out of 7,402 total goals (83.37%), and Passport had 6,352 explainable goals out of 7,994 total goals (79.46%). These results indicate that the majority of proof steps across all four provers can be explained through retrieved proofs, demonstrating that ProofFusion with the adaptive retrieval module significantly enhances the explainability of proof steps while improving their original proving capability. The consistently high proportion of explainable proof goals also suggests that the system can reliably provide understandable explanations for tactic selection across different provers. In addition, we further analyzed the reasons why some proof goals remain unexplainable, which will be discussed in detail in Section 5.1.

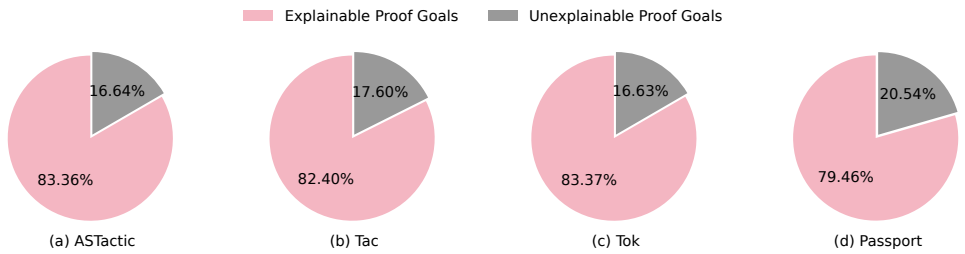


Fig. 8. Proportion of explainable proof goals across four neural theorem provers

Result of RQ5: ProofFusion significantly enhances the explainability of proof steps across all four neural provers, achieving explainable proof goal proportions of 83.36% for ASTactic, 82.40% for Tac, 83.37% for Tok, and 79.46% for Passport.

5 Discussion

In this section, we discuss the factors that cause ProofFusion to fail in providing explainable proof steps, as well as the potential threats to the validity of our study.

5.1 Understanding Limitations in Proof Goal Explainability

During proof search, we observed that some proof steps could not be clearly traced back to retrieval results. To investigate this phenomenon, we conducted a systematic analysis of these unexplained proof goals. The results indicate that the primary cause lies in the limitations of the generative model in producing candidate tactics. Specifically, although the number of retrievable tactics provided for each proof goal matches the number of candidate tactics generated by the model, only those candidates that align with retrieval results can be linked to historical proof goals after syntactic and category-based aggregation. Unmatched candidates are generally ranked lower. Consequently, when higher-ranked tactics fail to execute successfully in the proof assistant, the search process often relies on lower-ranked, unmatched tactics to advance the proof, resulting in unexplainable proof steps. In other words, when the generative model inadequately produces valid tactics, the system is forced to resort to tactics that do not match retrieval results to progress.

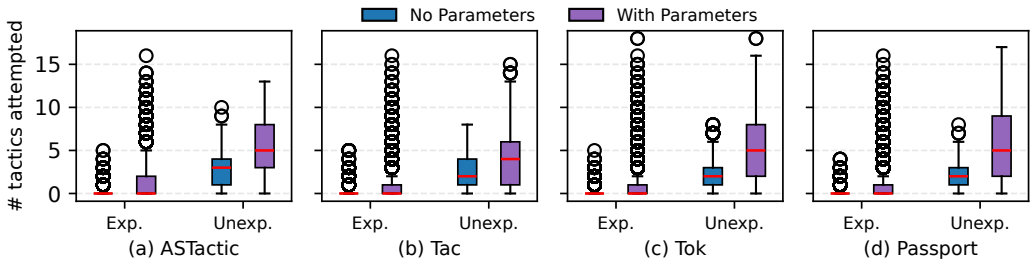


Fig. 9. Error distribution of attempted tactics in explainable and unexplainable proof goals

To further quantify this phenomenon, we measured the number of candidate tactics actually attempted in explainable and unexplainable proof steps. Candidate tactics were executed sequentially according to their ranking, interacting with Coq until a new proof goal was generated. As shown in Figure 9, unexplainable proof steps exhibit a significantly higher number of execution errors compared to explainable steps, with particularly elevated error rates for parameterized tactics. This indicates that even when candidate rankings are reasonable, deficiencies in the generation of tactic parameters of the generative model can lead to the production of numerous invalid tactics, causing all potentially explainable tactics to fail and ultimately forcing the system to rely on unexplainable tactics of lower rank.

In summary, these analyses reveal the direct impact of the generative model capabilities on the explainability of ProofFusion and suggest potential improvement directions, such as improving the accuracy and stability of parameter generation. Moreover, these findings imply that when constructing explainable neural theorem proving systems, relying solely on the retrieval module is insufficient to guarantee traceability. The execution stability of the generative model is also critical for ensuring system explainability.

5.2 Threats to Validity

5.2.1 Internal Validity. One potential threat to internal validity lies in the reliability of the neural theorem prover implementations we adopted. To mitigate this risk, we built our framework on top of their official open-source implementations and strictly relied on the publicly available codebases. Since our approach is an extension framework that does not require retraining the generation models, we directly used the publicly released checkpoints corresponding to the best-performing models and reran the original experiments to verify the results. In addition, when integrating our framework, we preserved the original model architectures and applied retrieval–generation

fusion only at the tactic level after generation, thereby avoiding any interference with the core implementations.

5.2.2 External Validity. A primary threat to external validity is whether the selected theorem proving dataset reflects the generalization capability of the proposed method. To address this, we used the widely adopted CoqGym dataset, which provides a real ITP environment for reliable evaluation under realistic conditions. A potential limitation is that CoqGym is based solely on the Coq proof assistant. Conceptually, ProofFusion can also apply to other tactic-based proof assistants such as Isabelle/HOL or Lean, which share two characteristics with Coq. Human-written proofs exhibit a long-tailed tactic distribution, and tactics generated by neural models often lack interpretability. Therefore, we expect ProofFusion could offer benefits in those systems as well. However, transferring our method to Isabelle/HOL or Lean is not straightforward because these systems differ in proof languages, tactic designs, state representations, and available corpora. As a result, all components, including proof goal extraction, tactic-aligned dataset construction, retriever training, and integration with the proof workflow, would need to be reimplemented and validated.

Another threat to external validity arises from the choice of neural theorem provers. To reduce this risk, we selected four representative models. While ProofFusion's effectiveness on other provers is yet to be validated, it improves the proof capabilities of all four selected provers, which helps mitigate the threat. Future work will extend ProofFusion to additional provers and datasets to further validate external validity.

6 Related Work

This section reviews related work on neural theorem proving, retrieval-augmented generation, and neural model enhancement.

6.1 Neural Theorem Proving

Automated proof tactics generation in ITP can be broadly categorized into two classes [20]. The first class is step-wise generation methods [2, 5, 12, 13, 51–53, 65], where the model generates candidate tactics at each step and incrementally constructs complete proofs through proof search, thereby learning proof strategies directly from data. The second class is large-model-driven approaches, which in principle, can also adopt a step-wise generation combined with search to leverage the reasoning capabilities of large models fully. However, due to the high cost of invoking large-scale language models, current research often resorts to a compromise: generating the entire proof in a single pass and relying on repair mechanisms to ensure correctness [14, 43]. While this approach reduces search overhead, it sacrifices flexibility and explainability in the proof generation process.

ProofFusion focuses on step-wise generation combined with search in neural theorem proving. Unlike existing studies that primarily design novel neural models, we aim to improve existing provers in a model-agnostic manner. We integrate semantic retrieval with generation, enhancing the explainability of tactics. ProofFusion can be seamlessly incorporated as a lightweight plugin into existing proof synthesis tools, improving proof success rates and tactic transparency without modifying the underlying models, providing an orthogonal complement to current techniques.

6.2 Retrieval-augment Generation

Retrieval-Augmented Generation (RAG) has been widely applied to code intelligence tasks. Prior studies have demonstrated its effectiveness in code generation and summarization [4, 16, 41, 48], program repair [19, 40, 42, 59], assertion generation [33, 56, 70, 71], and commit message generation [36, 54, 58, 63, 68]. In these tasks, RAG methods expand the input context of the model by retrieving external knowledge such as code repositories, repair patterns, or assertion examples,

thereby substantially improving performance. A common characteristic of these approaches is that the retrieved content is concatenated into the input of the model, enabling the generator to exploit external instances during decoding.

In contrast, applying RAG to formal verification and theorem proving presents unique challenges. Proof synthesis in interactive theorem proving requires generating sequences of logically valid tactics rather than natural language or code snippets. Although recent studies have explored retrieval for selecting related premise [66] or proof steps [57], they largely rely on lexical or generic semantic matching and still adopt input concatenation, which limits adaptability and explainability.

Departing from prior methods, ProofFusion makes two main contributions. First, we design a semantics-aware retriever that captures deep semantic relations between proof goals and candidate strategies, rather than relying on surface-level similarity. Second, we introduce a novel output-level probability re-ranking fusion mechanism, which moves beyond simple input concatenation, enhances adaptability to proof synthesis tasks, and provides stronger explainability.

6.3 Methods for Improving Neural Model Capabilities

In recent years, various approaches have been proposed to improve the capabilities of multiple neural network models under a unified framework. For example, MalTutor [32] and MalCertain [34] leverage predictive uncertainty to enhance the robustness of diverse neural networks in Android malware detection. GVI [67] generates diverse vulnerability samples with large language models, enhancing the generalization of multiple deep models in vulnerability detection. Coca [3] combines contrastive learning with dual-view causal reasoning, improving both robustness and explainability of various graph neural networks in GNN-based vulnerability detection. Legato [31] uses attention-guided graph augmentation and pseudo-label estimation to enhance the performance of multiple graph-structured models in fault localization.

Unlike these methods, which focus on discriminative tasks such as classification or detection, ProofFusion focuses on the generative task of automated theorem proving and provides a model-agnostic enhancement approach. ProofFusion improves both the capability and explainability of generative models without requiring retraining, offering a novel direction for performance optimization in generative tasks.

7 Couclusion

In this work, we present ProofFusion, an adaptive retrieval-augmented reasoning framework that enhances neural theorem provers without requiring retraining. By combining a proof semantic aware retriever with a dual-track reranking fusion mechanism, ProofFusion effectively integrates historical proof knowledge with neural model predictions, improving both the explainability and success rate of automated proof tactic generation. These results demonstrate that our approach functions as a practical and model-agnostic augmentation to existing neural theorem provers.

8 Data Availability

The dataset, source code, model checkpoints, and experimental results of this study are publicly available in a reproducibility package on Zenodo [69].

Acknowledgments

This work was supported by the Major Program of the National Natural Science Foundation of China (Grant Nos. 62192733 and 62192730), the General Program of National Natural Science Foundation of China No.62372219. We gratefully acknowledge the high-performance computing support provided by the Computing Center in Xi'an. We also sincerely thank the anonymous reviewers for their insightful and constructive comments.

References

[1] David Aspinall, Ewen Denney, and Christoph Lüth. 2010. Tactics for hierarchical proof. *Mathematics in Computer Science* 3, 3 (2010), 309–330.

[2] Lasse Blaauwbroek, Miroslav Olšák, Jason Rute, Fidel Ivan Schaposnik Massolo, Jelle Piepenbrock, and Vasily Pestun. 2024. Graph2Tac: Online representation learning of formal math concepts. *arXiv preprint arXiv:2401.02949* (2024).

[3] Sicong Cao, Xiaobing Sun, Xiaoxue Wu, David Lo, Lili Bo, Bin Li, and Wei Liu. 2024. Coca: Improving and explaining graph neural network-based vulnerability detection systems. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[4] Junkai Chen, Xing Hu, Zhenhao Li, Cuiyun Gao, Xin Xia, and David Lo. 2024. Code search is all you need? improving code suggestions with code search. In *Proceedings of the IEEE/ACM 46th international conference on software engineering*. 1–13.

[5] Yizhou Chen. 2025. Gpass: A Goal-Adaptive Neural Theorem Prover Based on Coq for Automated Formal Verification. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 653–653.

[6] Edmund M Clarke. 1997. Model checking. In *International conference on foundations of software technology and theoretical computer science*. Springer, 54–56.

[7] David Delahaye. 2000. A tactic language for the system Coq. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*. Springer, 85–95.

[8] Lihang Fan, Wenfei Fan, Ping Lu, Chao Tian, and Qiang Yin. 2023. Enriching recommendation models with logic conditions. *Proceedings of the ACM on Management of Data* 1, 3 (2023), 1–28.

[9] Feiteng Fang, Yuelin Bai, Shiwen Ni, Min Yang, Xiaojun Chen, and Ruifeng Xu. 2024. Enhancing Noise Robustness of Retrieval-Augmented Language Models with Adaptive Adversarial Training. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 10028–10039.

[10] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 1536–1547.

[11] Óscar Ferrández, Rafael Munoz Terol, Rafael Muñoz, Patricio Martínez-Barco, and Manuel Palomar. 2006. Deep vs. Shallow semantic analysis applied to textual entailment recognition. In *International Conference on Natural Language Processing (in Finland)*. Springer, 225–236.

[12] Emily First and Yuriy Brun. 2022. Diversity-driven automated formal verification. In *Proceedings of the 44th International Conference on Software Engineering*. 749–761.

[13] Emily First, Yuriy Brun, and Arjun Guha. 2020. TacTok: Semantics-aware proof synthesis. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–31.

[14] Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. 2023. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1229–1241.

[15] Melvin Fitting. 2012. *First-order logic and automated theorem proving*. Springer Science & Business Media.

[16] Xinyu Gao, Yun Xiong, Deze Wang, Zhenhan Guan, Zejian Shi, Haofen Wang, and Shanshan Li. 2024. Preference-guided refactored tuning for retrieval augmented code generation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 65–77.

[17] Herman Geuvers. 2009. Proof assistants: History, ideas and future. *Sadhana* 34, 1 (2009), 3–25.

[18] Hao Guo and Song Wang. 2021. Long-tailed multi-label visual recognition by collaborative training on uniform and re-balanced samplings. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 15089–15098.

[19] Sichong Hao, Xianjun Shi, and Hongwei Liu. 2024. RetypeR: Integrated Retrieval-based Automatic Program Repair for Python Type Errors. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 199–210.

[20] Jilin Hu, Jianyu Zhang, Yongwang Zhao, and Talia Ringer. 2025. HybridProver: Augmenting Theorem Proving with LLM-Driven Proof Synthesis and Refinement. *arXiv preprint arXiv:2505.15740* (2025).

[21] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. 1997. The coq proof assistant a tutorial. *Rapport Technique* 178 (1997), 113.

[22] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 7029–7043.

[23] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen Tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020*. Association for Computational Linguistics (ACL), 6769–6781.

[24] Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Nearest neighbor machine translation. *arXiv preprint arXiv:2010.00710* (2020).

[25] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172* (2019).

[26] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Advances in neural information processing systems* 33 (2020), 18661–18673.

[27] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1746–1751.

[28] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. 2009. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 207–220.

[29] Xavier Leroy. 2009. Formal verification of a realistic compiler. *Commun. ACM* 52, 7 (2009), 107–115.

[30] Xavier Leroy, Sandrine Blazy, Daniel Kästner, Bernhard Schommer, Markus Pister, and Christian Ferdinand. 2016. CompCert-a formally verified optimizing compiler. In *ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress*.

[31] Chun Li, Hui Li, Zhong Li, Minxue Pan, and Xuandong Li. 2024. Enhancing Fault Localization in Industrial Software Systems via Contrastive Learning. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 101–113.

[32] Haodong Li, Xiao Cheng, Yanjie Zhao, Guosheng Xu, Guoai Xu, and Haoyu Wang. 2025. Understanding Model Weaknesses: A Path to Strengthening DNN-Based Android Malware Detection. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 320–342.

[33] Hongyan Li, Weifeng Sun, Meng Yan, Ling Xu, Qiang Li, Xiaohong Zhang, and Hongyu Zhang. 2025. Retrieval-Augmented Fine-Tuning for Improving Retrieve-and-Edit Based Assertion Generation. *IEEE Transactions on Software Engineering* (2025).

[34] Haodong Li, Guosheng Xu, Liu Wang, Xusheng Xiao, Xiapu Luo, Guoai Xu, and Haoyu Wang. 2024. Malcertain: Enhancing deep neural network based android malware detection by tackling prediction uncertainty. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[35] Yikai Li, CL Philip Chen, and Tong Zhang. 2022. A survey on siamese network: Methodologies, applications, and opportunities. *IEEE Transactions on artificial intelligence* 3, 6 (2022), 994–1014.

[36] Zhihan Li, Yi Cheng, Haiyang Yang, Li Kuang, and Lingyan Zhang. 2022. Retrieve-guided commit message generation with semantic similarity and disparity. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 357–366.

[37] Zhaoyu Li, Jialiang Sun, Logan Murphy, Qidong Su, Zenan Li, Xian Zhang, Kaiyu Yang, and Xujie Si. 2024. A survey on deep learning for theorem proving. *arXiv preprint arXiv:2404.09939* (2024).

[38] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. 2356–2362.

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

[40] Pei Liu, Bo Lin, Yihao Qin, Cheng Weng, and Liqian Chen. 2024. T-RAP: A Template-guided Retrieval-Augmented Vulnerability Patch Generation Approach. In *Proceedings of the 15th Asia-Pacific Symposium on Internetware*. 105–114.

[41] Shangqing Liu, Yu Chen, Xiaofei Xie, Jingkai Siow, and Yang Liu. 2020. Retrieval-augmented generation for code summarization via hybrid GNN. *arXiv preprint arXiv:2006.05405* (2020).

[42] Zixin Liu, Xiaozhi Du, and Hairui Liu. 2025. ReAPR: Automatic program repair via retrieval-augmented large language models. *Software Quality Journal* 33, 3 (2025), 1–31.

[43] Minghai Lu, Benjamin Delaware, and Tianyi Zhang. 2024. Proof automation with large language models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1509–1520.

[44] Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2023. A Survey of Deep Learning for Mathematical Reasoning. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

[45] Filip Maric. 2015. A survey of interactive theorem proving. *Zbornik radova* 18, 26 (2015), 173–223.

[46] Hansa Meghwani, Amit Agarwal, Priyaranjan Pattnayak, Hitesh Laxmichand Patel, and Srikant Panda. 2025. Hard Negative Mining for Domain-Specific Retrieval in Enterprise Systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, Georg Rehm and Yunyao Li (Eds.). Association for Computational Linguistics, Vienna, Austria, 1013–1026. doi:10.18653/v1/2025.acl-industry.72

[47] M Saqib Nawaz, Moin Malik, Yi Li, Meng Sun, and M Lali. 2019. A survey on theorem provers in formal methods. *arXiv preprint arXiv:1912.03028* (2019).

[48] Md Rizwan Parvez, Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval Augmented Code Generation and Summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*. 2719–2734.

[49] Thilina C Rajapakse. 2023. Dense passage retrieval: Architectures and augmentation methods. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 3494–3494.

[50] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[51] Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. 2020. Generating correctness proofs with neural networks. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 1–10.

[52] Alex Sanchez-Stern, Emily First, Timothy Zhou, Zhanna Kaufman, Yuriy Brun, and Talia Ringer. 2023. Passport: Improving automated formal verification using identifiers. *ACM Transactions on Programming Languages and Systems* 45, 2 (2023), 1–30.

[53] Alex Sanchez-Stern, Abhishek Varghese, Zhanna Zhanna Kaufman, Dylan Zhang, Talia Ringer, and Yuriy Brun. 2024. QEDCartographer: Automating formal verification using reward-free reinforcement learning. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 405–418.

[54] Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. Race: Retrieval-augmented commit message generation. *arXiv preprint arXiv:2203.02700* (2022).

[55] Mark Staples, Ross Jeffery, June Andronick, Toby Murray, Gerwin Klein, and Rafal Kolanski. 2014. Productivity for proof engineering. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–4.

[56] Weifeng Sun, Hongyan Li, Meng Yan, Yan Lei, and Hongyu Zhang. 2023. Revisiting and improving retrieval-augmented deep assertion generation. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1123–1135.

[57] Kyle Thompson, Nuno Saavedra, Pedro Carrott, Kevin Fisher, Alex Sanchez-Stern, Yuriy Brun, João F. Ferreira, Sorin Lerner, and Emily First. 2025. Rango: Adaptive Retrieval-Augmented Proving for Automated Software Verification. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. 347–359.

[58] Haoye Wang, Xin Xia, David Lo, Qiang He, Xinyu Wang, and John Grundy. 2021. Context-aware retrieval-based deep commit message generation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 4 (2021), 1–30.

[59] Weishi Wang, Yue Wang, Shafiq Joty, and Steven CH Hoi. 2023. Rap-gen: Retrieval-augmented patch generation with codet5 for automatic program repair. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 146–158.

[60] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems* 33 (2020), 5776–5788.

[61] Xin-Cheng Wen, Cuiyun Gao, Feng Luo, Haoyu Wang, Ge Li, and Qing Liao. 2024. LIVABLE: exploring long-tailed classification of software vulnerability types. *IEEE Transactions on Software Engineering* 50, 6 (2024), 1325–1339.

[62] Shangyu Wu, Ying Xiong, Yufei Cui, Haolun Wu, Can Chen, Ye Yuan, Lianming Huang, Xue Liu, Tei-Wei Kuo, Nan Guan, et al. 2024. Retrieval-augmented generation for natural language processing: A survey. *arXiv preprint arXiv:2407.13193* (2024).

[63] Bo Xiong, Linghao Zhang, Chong Wang, and Peng Liang. 2025. Contextual Code Retrieval for Commit Message Generation: A Preliminary Study. *arXiv preprint arXiv:2507.17690* (2025).

[64] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808* (2020).

[65] Kaiyu Yang and Jia Deng. 2019. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning*. PMLR, 6984–6994.

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

[67] Heng Yong, Zhong Li, Minxue Pan, Tian Zhang, Jianhua Zhao, and Xuandong Li. 2025. GVI: Guided Vulnerability Imagination for Boosting Deep Vulnerability Detectors. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 750–750.

[68] Linghao Zhang, Hongyi Zhang, Chong Wang, and Peng Liang. 2024. Rag-enhanced commit message generation. *arXiv preprint arXiv:2406.05514* (2024).

[69] Manqing Zhang, Yunwei Dong, Lingru Zhou, Bingxu Xiao, and Yepang Liu. 2025. Replication Package for ProofFusion. https://doi.org/10.5281/zenodo.17096183.

[70] Quanjun Zhang, Chunrong Fang, Yi Zheng, Ruixiang Qian, Shengcheng Yu, Yuan Zhao, Jianyi Zhou, Yun Yang, Tao Zheng, and Zhenyu Chen. 2025. Improving Retrieval-Augmented Deep Assertion Generation via Joint Training. *IEEE Transactions on Software Engineering* (2025).

[71] Quanjun Zhang, Chunrong Fang, Yi Zheng, Yaxin Zhang, Yuan Zhao, Rubing Huang, Jianyi Zhou, Yun Yang, Tao Zheng, and Zhenyu Chen. 2025. Improving Deep Assertion Generation via Fine-Tuning Retrieval-Augmented Pre-trained Language Models. *ACM Transactions on Software Engineering and Methodology* (2025).

[72] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. 2024. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473* (2024).

[73] Zhisheng Zhong, Jiequan Cui, Shu Liu, and Jiaya Jia. 2021. Improving calibration for long-tailed recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 16489–16498.

Received 2025-09-11; accepted 2025-12-22