PROOFWALA: A Framework for Multilingual Proof Data Synthesis and Theorem-Proving

Amitayush Thakur [1] **George Tsoukalas** [1] **Greg Durrett** [1] **Swarat Chaudhuri** [1]

Abstract

Neural networks have shown substantial promise at automatic theorem-proving in interactive proof assistants (ITPs) like Lean and Coq. However, most neural theorem-proving models are restricted to specific ITPs, leaving out opportunities for cross-lingual *transfer* between ITPs. We address this weakness with a multilingual proof framework, PROOFWALA, that allows a standardized form of interaction between neural theorem-provers and two established ITPs (Coq and Lean). It enables the collection of multilingual proof step data—data recording the result of proof actions on ITP states—for training neural provers. PROOFWALA allows the systematic evaluation of a model's performance across different ITPs and problem domains via efficient parallel proof search algorithms. We show that multilingual training enabled by PROOFWALA can lead to successful transfer across ITPs. Specifically, a model trained on a mix of PROOFWALA-generated Coq and Lean data outperforms Lean-only and Coq-only models on the standard prove-at-k metric. We open source all our code, including PROOFWALA parallel proof search framework, the Multilingual ITP interaction framework. , and models PROOFWALA-MULTILINGUAL, LEAN, and COQ. Our full dataset is also available on HuggingFace.

1. Introduction

Automated theorem-proving has long been considered to be a grand challenge in artificial intelligence. Recently, deep learning has emerged as a promising approach to this challenge (Li et al., 2024; Yang et al., 2024). Broadly, deep-learning methods for theorem-proving use neural models to generate formal proof expressed in the language of an *interactive theorem prover* (ITPs), e.g., LEAN (de Moura et al., 2015), COQ (Huet et al., 1997), or Isabelle (Paulson, 1994). An ITP represents proofs as sequences of simplification steps, or *tactics*, and can mechanically check such proofs for correctness. Theorem-proving then amounts to generating a sequence that passes the ITP's checks.

Most deep-learning approaches to theorem-proving follow the strategy proposed by Polu & Sutskever (2020). Here, one first trains a generative language model (LM) that can predict formal proof steps (tactics and their parameters) conditioned on the goal state, from a proof-step dataset extracted from existing formal mathematics repositories. The learned model is then wrapped in a search algorithm which conducts proof search (see Section 2 for more details).

While neural approaches to theorem-proving are gaining momentum, the field remains fragmented. Existing tools for dataset collection tend to be ITP-specific, often relying on isolated, domain-specific formats; there is also no language-agnostic open platform for neurally guided search over proofs. This hinders systematic comparisons and precludes potential cross-lingual and cross-domain improvements from training on multilingual data.

In response to this problem, we introduce PROOFWALA[2], a multilingual framework for dataset collection, interaction, training, and proof search across interactive theorem provers and domains. PROOFWALA provides a standardized pipeline for generating proof step training data, facilitating the creation of high-quality multilingual datasets. It enables seamless interaction with formal systems and supports the training of neural architectures tailored for proof step generation. Finally, it integrates efficient search algorithms, including a parallelized version of best-first and beam search, allowing for end-to-end proof discovery guided by transformer-based models.

[1]Department of Computer Science, University of Texas, Austin, USA. Correspondence to: Amitayush Thakur <amitayush@utexas.edu>, George Tsoukalas <george.tsoukalas@utexas.edu>, Greg Durrett <gdurrett@cs.utexas.edu>, Swarat Chaudhuri <swarat@cs.utexas.edu>.

The second AI for MATH Workshop at the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. Non-archival.

[2]"Wala" is a suffix from Indic languages (often spelled "wallah"), meaning "one who is associated with or provides a particular thing."

We provide a code library combined with multilingual datasets and multilingual fine-tuned models that facilitate end-to-end formal proof search in LEAN 4 and COQ. Using PROOFWALA, we demonstrate that training on multilingual data can foster positive cross-lingual and cross-domain transfer, enhancing proof generation across different formal systems.

In summary, our work makes three key contributions:

A Standardized Framework: We propose PROOFWALA, a unified framework for extracting and organizing training data for formal theorem proving in LEAN and COQ. The framework supports the generation of training data for these ITPs from any formal theorem-proving Git repository (such as Mathlib, CompCert, MathComp, GeoCoq, & Category Theory) and the subsequent training of LLMs for single proof step generation. Our data collection format is standardized across ITPs, and we have created generic prompt formatting schemes across different ITPs and domains. The framework also helps with end-to-end proof search and collection of the annotated proof trees which can be further used for analysis and visualization (see Figure 7 in Appendix B.5). All our code is open source, the PROOFWALA framework can be found here. The code for the multilingual ITP interaction module can be found here.

Support for Parallel Poof Completion: Similar to Polu & Sutskever (2020), the framework supports proof completion using a search guided by the proof step generation model. We improve the search by parallelizing it and making it agnostic of the ITP. To the best of our knowledge, ours is the first open-source framework that supports **parallel proof-search** by adding capabilities to clone proof environments and run tactics in parallel across those proof environments. We further build capabilities to **store, annotate, and visualize the proof trees generated during the search**. Figure 7 in Appendix B.5 shows the visualization of the proof-tree generated during the proof search.

Demonstration of Cross-Lingual and Cross-Domain Transfer: Facilitated by our PROOFWALA framework, we investigate the effect of incorporating multilingual proof data in the training pipeline. We demonstrate that **cross-domain and cross-lingual transfer** occur for both LEAN and COQ, in both the domains of general mathematics and software verification. These results highlight the potential of training across diverse formal proof assistant repositories as an effective strategy to mitigate data scarcity in this neural theorem-proving research. We release the multi-domain and multi-lingual training data used for training our models, containing about 450K training data points (270M tokens) from about 80k theorems. We also release multiple PROOFWALA models trained on different data-mixes. Our PROOFWALA-MULTILINGUAL model is the *first open proof step generation model* trained on data from diverse domains and ITPs, which can be seamlessly used for finding proofs in formal mathematics and software verification.

2. Problem Formulation

We view a theorem-prover as a system that systematically addresses a set of *proof obligations* by applying a sequence of proof tactics. Each obligation o is a pair (g, h), where g is the goal to be proven and h contains the hypotheses relevant to proving g. The system starts with an initial set of proof obligations; its ultimate goal is to reduce this set to an empty set.

Figure 1 shows a formal proof of a theorem about block triangular matrices —found using the PROOFWALA-MULTILINGUAL proof-step generation model—in the LEAN 4 language (de Moura et al., 2015).

As in Thakur et al. (2024), we treat theorem-proving as a discrete search through the state space of an ITP. We abstractly model an ITP as a *proof environment* consisting of a set of *states* $\mathcal{O}$, where each *state* is a set $O = \{o_1, \ldots, o_k\}$ of obligations o_i. The *initial state*, $\mathcal{I}$, consisting of a single obligation (g_{in}, h_{in}) extracted from a user-provided theorem. A unique *goal state* `QED` is the empty obligation set. A finite set of *proof tactics*. A transition function $T(O, a)$, which determines the result of applying a tactic a to a state O. If a can be successfully applied at state O, then $T(O, a)$ is the new set of obligations resulting from the application. If a tactic a cannot be applied to the state O, then $T(O, a) = O$. We define the transition function T_{seq} over a sequence of proof-steps (tactics), $\alpha = \langle a_1, a_2, \ldots, a_n \rangle$, and proof state, $O \in \mathcal{O}$, as:

$$T_{seq}(O, \alpha) = \begin{cases} T(O, a_1) & \text{if } n = 1 \\ T(T_{seq}(O, \langle a_1, \ldots, a_{n-1} \rangle), a_n) & \text{otherwise.} \end{cases}$$

The theorem-proving problem is now defined as follows:

Problem 1 (Theorem-proving) *Given an initial state O_{in} find a tactic sequence α (a* proof*) satisfying $T_{seq}(O_{in}, \alpha) =$* `QED`.

Proving with LMs. Our proof search approach reflects the strategy described in (Polu & Sutskever, 2020), where a neural model predicts a tactic to apply given the current state, namely $p(a|O)$. Such a model can be implemented with a language model (LM) that generates a tokenized representation of a tactic a in a token-by-token fashion: $p(\text{tok}(a)|O)$. For simplicity of notation, we drop references to tokenization in the remainder of the paper, but notably, statements from different ITPs can be tokenized into a shared LM vocabulary. The learned model p is the workhorse of our multilingual prover, which we will train using multilingual proof data.

```
theorem blockTriangular_stdBasisMatrix
{i j : m} (hij : b i ≤ b j) (c : R)
: BlockTriangular (stdBasisMatrix i j c) b
:= by rintro i' j' hij'
  simp [stdBasisMatrix, hij, hij'.not_le]
  rintro rfl rfl
  exact (not_lt_of_le hij hij').elim
```

Figure 1: A LEAN 4 theorem and a with a correct proof using PROOFWALA-MULTILINGUAL proof-step generation model. The theorem states that the standard basis matrix, where c is placed in the $(i, j)th$ entry with zeroes elsewhere is block triangular. The first tactic `rintro i' j' hij'` unfolds the definition of `BlockTriangular` and adds the variables `i'`, `j'`, as well as the hypothesis `hij' : b j' < b i'` to the set of hypotheses. The proof proceeds by using established properties of the `stdBasisMatrix` and resolves by demonstrating an inconsistency with the hypothesis `hij : b i ≤ b j`.

3. Framework Details

Now we describe the PROOFWALA framework. Our main motivation for building this new framework is to support theorem-proving research in a *language-agnostic* manner. In particular, we aim to facilitate standardized data collection in different ITPs and provide the necessary infrastructure to train proof step generation models, along with efficient parallel search algorithms for proof synthesis conditioned on theorem specifications.

Our framework has three components: (i) the **interface module**: for the execution of proof steps (tactics) on various ITPs, (ii) the **proof step generation & training module**: for generating proof step data and training proof step generation model, and (iii) the **parallel proof search module**: for using the guidance from proof step generation model to do end-to-end the proof search. Figure 2 shows the interaction between our different modules.

3.1. Interface Module

First, we detail the interface module, which is responsible for facilitating interaction with the ITPs when executing proof steps. In particular, the interface module supports interaction with LEAN 4 and COQ (multiple versions from 10.0 - 18.0). Our COQ implementation is built on top of `coq_serapy`[3] (Sanchez-Stern et al., 2020), while our LEAN 4 implementation is built on top of the `REPL`[4] library. Notably, neither of these libraries has the capability to do parallel interactions with ITPs. Hence, we created a pooling mechanism that allows us to make multiple instances of the interface module with the same state to execute tactics in parallel (parallelizable across multiple machines on a Ray cluster) for the same theorem. Parallelism is essential for searching for proofs or annotating proofs found at scale. We also fixed some well-known bugs and limitations with these libraries (see Appendix B.4). Our abstraction can support any future versions of Lean and Coq since we use the language server protocol (LSP) to further abstract out the low-level interaction between our code and the ITP interpreter/compiler.

One challenge in creating a unified framework is supporting the variety of state representations across these different ITPs. We develop a standard representation consistent with our problem formulation in Section 2 that is generic enough to cover all supported ITPs. The collected data is stored as `json` in the unified format across different ITPs; Figure 4 in Appendix A.1 shows the generalized format used for collecting training data.

While our interface abstraction generalizes across Lean and Coq, we also extended preliminary support to Isabelle via the PISA server. However, due to substantial resource overheads—such as high memory and disk usage per PISA instance—we do not officially support Isabelle in our large-scale experiments. These limitations are discussed in detail in Appendix B. Nonetheless, our data formats and interaction logic remain compatible, and we view full Isabelle integration as a promising direction once more scalable tooling becomes available.

3.2. Proof Step Generation and Training Module

Next, we describe our dataset collection & training module, which is designed to support the production of the proof step prediction model $p(a|O)$. The first step in training a proof step generation model is to extract (proof state, proof step) pairs from human written proofs in various repositories. We use our **interface module** (Section 3.1) to interact with the ITP and collect proof state and proof step (tactic) pair data from all theorems in a given formal proof repository such as COMPCERT, MATHLIB, etc. For a given theorem statement and its corresponding formal proof, we extract the sequence of tactics $\alpha = \langle a_1, a_2, \dots a_n \rangle$ and their corresponding state transitions. Namely, for each theorem in the repository, we extract the sequence of pairs $\pi = \langle (O_0, a_1), \dots, (O_{i-1}, a_i), \dots (O_{n-1}, a_n) \rangle$, such that $O_0 = \{(g_{in}, h_{in})\}$ (extracted from the theorem statement itself), $T(O_i, a_i) = O_{i+1}$, and $T(O_{n-1}, a_n) =$ `QED`. Apart from collecting the current state, proof step, and the next state, we also collect information about other lemmas which are referenced in the proof step. Figure 4 in Appendix A.1 shows the data extracted for a theorem in COQ and LEAN 4.

PROOFWALA includes functionality for training neural

[3] https://github.com/HazardousPeach/coq_serapy
[4] https://github.com/leanprover-community/repl

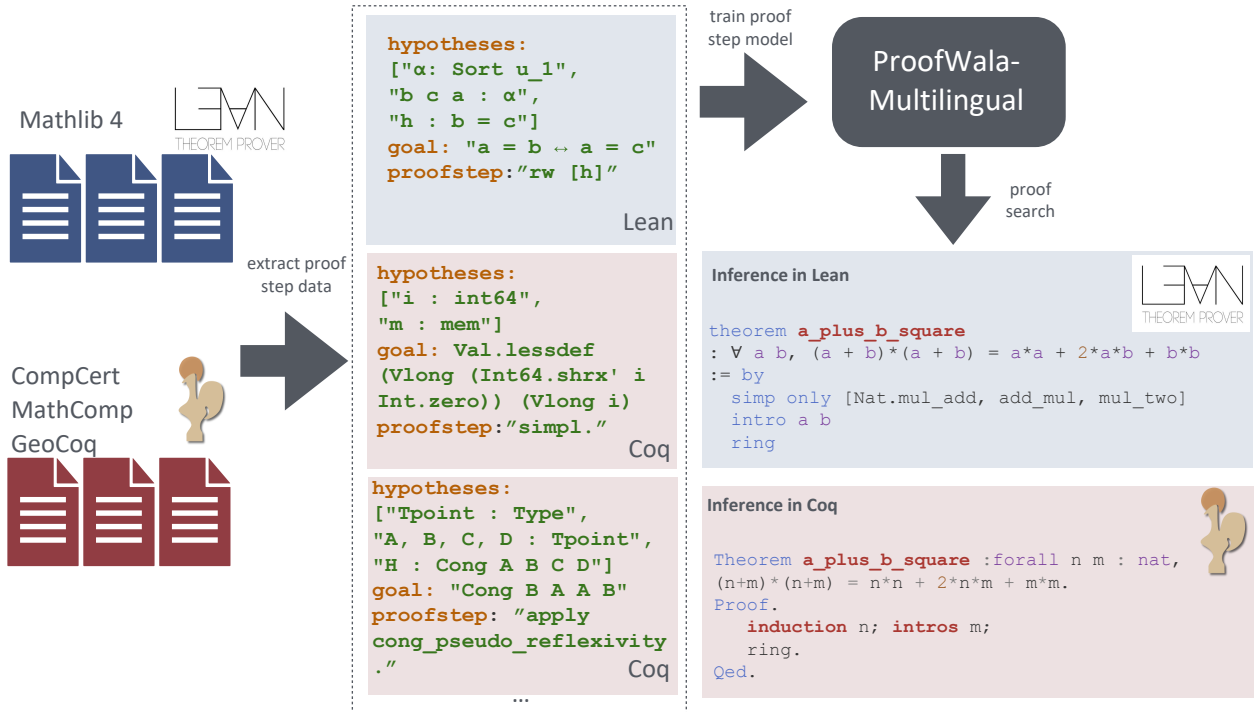

Figure 2: The PROOFWALA Framework with the interaction between different modules. Using PROOFWALA's interaction & data-collection modules, we collect a multilingual proof dataset from existing formal mathematics repositories in LEAN and COQ. The resulting dataset is used to train a multilingual proof step prediction model, supported by PROOFWALA's training module. The multilingual model is used inside PROOFWALA's search module to conduct proof search.

models on the constructed proof datasets. It supports fine-tuning any pretrained HUGGINGFACE model for proof step generation using the data extracted from the formal proof repositories. We support generic yet flexible input formats (prompt formats) for supervised fine-tuning of the language model to predict the next proof steps. The prompt is standardized across languages and different versions of ITP and controls what aspects of the state are used for predicting the next proof step. Figure 5 in Appendix A.1 shows the example prompt formats used for training. Our format is inspired by COPRA (Thakur et al., 2024) but does not use error signals. To allow transfer across different ITPs, we do not provide any information about the domain or ITP assistant that produced the state mentioned in the prompt. As an example, we choose CODET5-BASE (Wang et al., 2021) as our pretrained model for fine-tuning in our experiments.

3.3. Parallel Proof Search Module

The proof search module uses the proof step generation model, trained via the **proof step generation and training module** (see Section 3.2), to direct the proof search through the sampling of possible next proof steps for a given state. In particular, the purpose of the search module is to generate the sequence of proof steps (tactics) $\alpha = \langle a_1, a_2, \ldots a_{n-1} \rangle$ and sequence of proof-states $\omega = \langle O_0, O_1, \ldots O_n \rangle$ where (i) given a proof state O, we draw N samples from the proof step generation model to get a set $\mathbb{A}(O) = \{a_1, \ldots, a_k\}$ of possible proof steps, (ii) and for each $i \in [n-1]$ there exists $a_{i+1} \in \mathbb{A}(O_i)$ such that $T(O_i, a_{i+1}) = O_{i+1}$ and (iii) the final state is `QED`: $T(O_{n-1}, a_n) = O_n = \texttt{QED}$. The proof search module can support any custom tree search algorithm by abstracting the node selection, generation, and expansion logic. We implement beam search and best first search.

We maintain an annotated proof tree while searching for the sequence of proof step(s), α, which completes the proof for a given theorem. A fully annotated proof tree is shown in Figure 7 in Appendix B.5, which was generated while performing the beam search for proving a modulo arithmetic problem. We also use these trees to analyze the proofs generated by our models (see Section 5.1). We use the negative log-likelihood of the tokens generated by the PROOFWALA models for deciding the node expansion order in our proof search experiments.

Unlike other frameworks, our proof search module can run a parallel beam search using Ray (Moritz et al., 2018) for a given theorem. For example, frameworks like LeanDojo (Yang et al., 2023) for LEAN 4 searches for proofs sequentially for a given theorem. Parallel search improves our

throughput by trying to execute multiple possible proof step(s) (tactics) generated by PROOFWALA models in parallel on the ITP. We achieve this by replicating instances of **interface module** (see Section 3.1) into a custom pool of *Ray actors*. The custom pool tracks ITP instances' proof states and uses only those matching the frontier state (states that are being explored during search) to continue exploration, adding instances as needed. The search picks up multiple instances from this pool to execute the possible next proof step generated in parallel, hence avoiding the cost of sequentially running those steps one after another on the same instance of ITP. Figure 6 (in Appendix B.1) describes the pseudocode for parallel beam search as supported by this module. The parallel proof search module allows our framework to scale to proof search for more challenging theorems with better efficiency in a generic way.

4. Dataset and Model Details

In this section, we explain our dataset construction and model training choices towards our demonstration of positive transfer from multilingual training as well as adaptability to new domains via further fine-tuning.

4.1. Dataset Details

We collect datasets across multiple languages and language versions of Coq and Lean 4, sourcing data from existing repositories. Our data collection approach involves collecting proof states from the ITP through tactic execution. We construct several data-mixes, of different subsets of the accumulated data, to train various monolingual and multilingual PROOFWALA models to perform proof step prediction. The training data is formatted into prompts as shown in Figure 5 (in Appendix A.1). We collect proof-step data for the various data mixtures as shown in Table 1.

We use different Coq and Lean repositories to generate this proof-step data. We use well-known repositories, namely CompCert, Mathlib, MathComp, GeoCoq, and CategoryTheory, to generate the proof-step data. For CompCert we used the train-test split proposed by Sanchez-Stern et al. (2020), and for Mathlib we used the split proposed by Yang et al. (2023). Together we have `442607` proof-step pairs derived from over `76997` theorems across Lean and Coq (details of the split shown in Table 2). We hold out the CategoryTheory dataset from initial training data-mixes for experimentation with further fine-tuning for our novel domain adaptation experiment.

[4]Same as Proverbot split (Sanchez-Stern et al., 2020)

[5]Same as random split in ReProver (Yang et al., 2023)

Initial Fine-tuning			
Data-mix	**Data-mix Source**	**PROOFWALA Models Trained**	**Token Count**
1. CompCert	CompCert Repo[4]	-	61.6 M
2. MathComp	MathComp Repo	-	18.2 M
3. GeoCoq	GeoCoq Repo	-	91.2 M
4. COQ	**Data-Mixes:** 1-3	COQ	171 M
5. LEAN	Mathlib Repo[5]	LEAN	99 M
6. MULTILINGUAL	**Data-Mixes:** 4-5	MULTILINGUAL	270 M
Further Fine-tuning			
7. CategoryTheory	CategoryTheory Repo	MULTILINGUAL-CAT-THEORY & COQ-CAT-THEORY	1.7 M

Table 1: Different data-mixes used to extract proof-step and proof state pair data. Various PROOFWALA models trained on these data mixes.

4.2. Model Details

We used the CODET5-BASE (Wang et al., 2021) pretrained model—which has `220` million parameters—to fine-tune models on the different data-mixes as described in Table 1. We trained three models PROOFWALA-{MULTILINGUAL, COQ, LEAN} with the same step count and batch sizes for all settings. Training the models with the same number of steps aligns with recent work on training models for multilingual autoformalization (Jiang et al., 2023) which ensures that each model has the same number of gradient updates. Our models are initially trained on CompCert, Mathlib, MathComp, and GeoCoq. The hyperparameters used for training are described in Table 4 in Appendix B.2.

We selected CODET5 as the base model to balance our research objectives with computational constraints. Our goal was to investigate the benefits of multilingual training for theorem proving, which required training multiple models (see Tables 1 and 3) across diverse ITPs using over 400k proof-step-state pairs. Larger models like DeepSeek-Coder or Qwen-Math would have significantly increased the computational cost, making such extensive experimentation infeasible within our budget. Despite CODET5 being smaller than recent SoTA code models, it was sufficient to demonstrate meaningful cross-lingual transfer effects—highlighting that gains were not simply due to increased parameter count or data volume, but genuine structural generalization across formal languages.

To demonstrate the usefulness of our models on subsequent theorem-proving tasks, we perform further fine-tune of our PROOFWALA-{MULTILINGUAL, COQ} models on CategoryTheory theory data. We used the same hyperparameters

[6]While the LeanDojo dataset (Yang et al., 2023) officially has 2000 test theorems, only 991 of these are proved using tactics and have their tactics extracted in the dataset. Since our approach involves generating only tactic-based proofs, our Lean dataset is collected from those theorems with tactic-based proofs.

	# Proof-Step & State Pairs			Theorem Count		
Data-mix	**Train**	**Test**	**Val**	**Train**	**Test**	**Val**
1. CompCert	80288	6199	-	5440	501	-
2. MathComp	34196	1378	2285	11381	536	729
3. GeoCoq	91120	12495	4928	4036	505	208
4. COQ	205604	20072	7213	20857	1542	937
5. LEAN	237003	4323	4220	56140[6]	991[6]	1035[6]
6. MULTILINGUAL	442607	24395	11433	76997	2533	1972
7. CategoryTheory	4114	610	208	573	101	43

Table 2: Size of different data-mixes. The PROOFWALA models were trained on the training split of COQ, LEAN, and MULTILINGUAL data-mixes. After extracting proof-step and state pair data, random training, validation, and test splits are constructed with at least 500 test theorems except for CategoryTheory. For the LEAN and CompCert data-mix we used the same split as proposed by Yang et al. (2023)[6] and Sanchez-Stern et al. (2020) respectively.

as Table 4 (in Appendix B.2) but we reduce the number of training steps to 1200 and batch size to 8.

5. Evaluation

Using our trained PROOFWALA models, we investigate (i) the benefit of incorporating multilingual data into the training pipeline (ii) moreover, whether further fine-tuning multilingual models demonstrates superior adaptation to novel domains.

In particular, we use the PROOFWALA-{MULTILINGUAL, COQ, LEAN} models inside the search module afforded by our framework (see Section 3.3). Our experiments run proof search on the test split mentioned in Table 2 for the CompCert, MathComp, GeoCoq, CategoryTheory, and LEANdata-mixes. This enables us to study the impact of transfer in the case of the PROOFWALA-MULTILINGUAL model for diverse ITPs and domains.

5.1. Experiments

Setup. All our experiments use the PROOFWALA proof step models for single-step prediction and then use PROOFWALA to conduct proof search. In our experiments we employ beam search as the search algorithm. We use the negative log-likelihood of the tokens generated by the PROOFWALA proof step prediction model to direct the search. Figure 7 in Appendix B.5 shows one such search result. Hyperparameters used in our search algorithm are listed in Table 5 in Appendix B.3. We employ a timeout of `600` seconds for most of our experiments. However, for the GeoCoq data-mix, we set a higher timeout of `1200` seconds to accommodate the appreciably longer ground-truth proofs, which require more time to execute all generated proof steps.

[7]The results are statistically significant using a paired bootstrap test if $p_{\text{value}} < 0.05$.

We conduct ablations to study the impact of training PROOFWALA models on different data-mixes. We also run paired bootstrap hypothesis testing to better understand the significance of transfer happening between different data-mixes, and whether PROOFWALA-MULTILINGUAL has a significant edge over other monolingual models (PROOFWALA-COQ and PROOFWALA-LEAN) while searching for proofs.

Aggregate Results. Table 3 summarizes *pass@k* results ($1 \leq k \leq 5$) across all data-mixes—LEAN, CompCert, MathComp, GeoCoq, and CategoryTheory—using various PROOFWALA models. PROOFWALA-MULTILINGUAL consistently outperforms the monolingual variants (PROOFWALA-COQ and PROOFWALA-LEAN), and surpasses the prior SoTA (Proverbot (Sanchez-Stern et al., 2020)) on the CompCert dataset.

Paired bootstrap significance testing confirms that these improvements are statistically significant on the largest data-mix (Mathlib/LEAN), while other gaps are either smaller or based on limited test sets. Overall, the multilingual model offers superior proof search capabilities compared to single-ITP models.

To assess generalization, we fine-tuned PROOFWALA-MULTILINGUAL and PROOFWALA-COQ on CategoryTheory data. As shown in Table 3, the multilingual variant outperformed the Coq-only model by nearly 8%, highlighting improved adaptability to new domains. This suggests that multilingual training, especially when combined with task-specific fine-tuning, is more effective for assisting emerging formal repositories.

Additional Benchmarking and Scalability Analysis. We further evaluated our models on the `MiniF2F` benchmark. PROOFWALA-MULTILINGUAL achieved a `pass@5` of 26.23%, exceeding PROOFWALA-LEAN's 25.41%. This reinforces our broader findings on multilingual training benefits.

To study scalability, we varied the number of CPUs during parallel search. Increasing parallelism from 8 to 20 CPUs improved MiniF2F `pass@5` from 22.54% to 26.23%, while reducing average proving time from 83.32s to 74.56s. These gains arise from faster exploration and earlier pruning of unpromising paths, demonstrating the practical impact of our parallel search infrastructure.

Cross-Lingual Transfer in Category Theory. On the CategoryTheory benchmark, PROOFWALA-MULTILINGUAL-CAT-THEORY surpassed the Coq-only variant by 8%—the largest observed gain in our study. This improvement stems from strong cross-lingual transfer: many Lean theorems in Category Theory (e.g., involving `uncurry`, `counit`, `adjunction`) resemble Coq counterparts. Our multilingual model succeeded on Coq theorems that had analogs in its

Data-Mix			Pass-at-k %					
Name	**# Theorems**	**Proof Step Model**	**Pass@1**	**Pass@2**	**Pass@3**	**Pass@4**	**Pass@5**	p_{value} (α: 0.05)[7]
LEAN	991	PROOFWALA-LEAN	24.92	26.64	27.54	28.05	28.25	
		PROOFWALA-MULTILINGUAL	**26.84**	**28.56**	**29.67**	**29.97**	**30.58**	**0.018**
MathComp	536	PROOFWALA-COQ	**28.28**	28.65	29.4	29.59	30.15	
		PROOFWALA-MULTILINGUAL	27.9	**29.21**	**29.59**	**30.15**	**30.52**	0.355
GeoCoq	505	PROOFWALA-COQ	**32.87**	**33.66**	33.86	34.06	34.46	
		PROOFWALA-MULTILINGUAL	30.89	**33.66**	**34.65**	**35.64**	**35.84**	0.135
CompCert	501	PROOFWALA-COQ	17.56	18.76	19.16	19.76	20.76	
		PROOFWALA-MULTILINGUAL	**17.96**	**19.76**	**20.56**	**21.16**	**21.96**	0.191
CategoryTheory	101	PROOFWALA-COQ-CAT-THEORY	36.63	42.57	44.55	44.55	45.54	
		PROOFWALA-MULTILINGUAL-CAT-THEORY	**44.55**	**51.49**	**52.48**	**53.47**	**53.47**	**0.008**

Table 3: Comparison between various PROOFWALA models and the PROOFWALA-MULTILINGUAL model on different data-mixes. We can see that transfer happening between Lean and Coq on all data-mixes from various domains in math and software verification. We observe that the MULTILINGUAL model outperforms the LEAN and COQ models on all data mixes. The performance improvement is also statistically significant on the biggest data-mix LEAN (Mathlib). We also observe that after further fine-tuning, the MULTILINGUAL model significantly outperforms the COQ model on the CategoryTheory dataset.

Lean training data, whereas the monolingual model failed. Table 6 in Appendix C presents concrete examples of such theorem pairs, highlighting cases where only the multilingual model found a valid proof.

Interestingly, this contradicts common patterns in multilingual NLP. For example, XLM-R (Conneau et al., 2019) shows that multilingual training can lead to negative transfer in low-resource settings, where out-of-language tokens dilute performance. In contrast, we observe that training on mixed-language proof data ($M > N$ tokens) yields genuine gains over smaller monolingual datasets (N tokens), suggesting that formal languages offer stronger structural alignment for transfer.

Analysis of Specific Proofs. To further investigate multilingual benefits, we analyzed search trees constructed during proof attempts (*proof trees*; see Figure 7 in Appendix B.5). These trees include only compilable edges, ensuring each node corresponds to a valid state.

We find that proof trees generated by multilingual models have more nodes and edges (Table 7), indicating broader exploration. Figures 8 and 9 (in Appendix C.1) illustrate these trends. Moreover, multilingual models utilize the full timeout more often, while monolingual models frequently stall early. Table 8 and fig. 11 show time comparisons.

Multilingual models also produced higher-degree nodes (Figure 3), suggesting a greater diversity of correct, compilable proof steps per state. They often found multiple distinct proofs for a single theorem (Figure 10). These results reflect the enhanced search capacity of multilingual models and underscore the utility of our framework in analyzing such patterns.

6. Related Work

Previous open-sourced tooling has been developed for interaction with formal proof assistants, but individually only using a single language. Oftentimes, this tooling also contains data extraction features, compiling proof datasets from popular formalization repositories such as Mathlib (mathlib Community, 2020) for Lean, CompCert (Leroy, 2009) and Mathcomp (Mathcomp, 2015) for Coq. LeanDojo (Yang et al., 2023) provided open-source tooling for interaction with Lean 3 and extracted a proof step dataset from Mathlib.[8] NTP Toolkit (Zhu et al., 2023) supports extracting training data from arbitrary Lean repositories. CoqGym (Yang & Deng, 2019) is a framework for interaction and data collection with Coq up to versions 8.12.0 (because of dependency on SerAPI library[9]). Proverbot (Sanchez-Stern et al., 2020) introduced Coq-Serapy, an interaction tool in Coq from which our Coq support is derived. CoqPyt (Carrott et al., 2024) is a framework for interaction and data generation from Coq with emphasis on supporting LM-based methods. COPRA (Thakur et al., 2024) introduces a framework for interaction with Lean 3 and Coq, but without tooling for data extraction or support for heavy parallelism during proof search. Aniva et al. (2024) introduced Pantograph, an interaction and data collection framework for Lean 4. We remark that one of our main contributions is an *unified* framework for interacting with and collecting data from both Coq and Lean 4, with support for training and parallel search, hence affording automated theorem-proving researchers a common tool in the presence of multiple popular proof assistant languages.

A number of proof search methodologies have been pro-

[8] LeanDojo now supports Lean 4, which is not backwards compatible to Lean 3.

[9] https://github.com/rocq-archive/coq-serapi

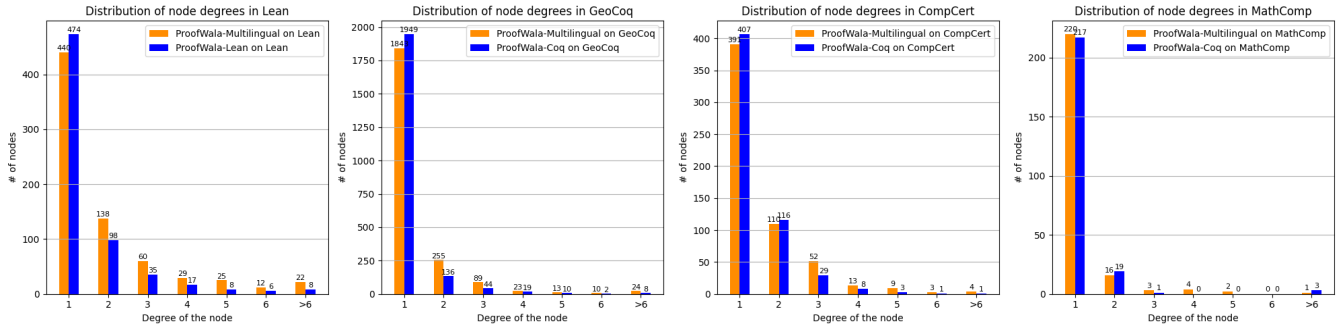

Figure 3: Distribution of degree of nodes in the proof trees across various data mixes found by different PROOFWALA models. Across all data mixes, PROOFWALA-MULTILINGUAL models tend to have higher degrees per node. This indicates that PROOFWALA-MULTILINGUAL often find more compilable tactics given a particular proof state, this can increase the chances of eventually finding a proof.

posed in the recent literature. GPT-f (Polu & Sutskever, 2020) employed a best-first search approach with a trained transformer-based architecture for proof synthesis. LeanDojo (Yang et al., 2023) similarly employs a best-first search, though augments the neural prediction model with a retrieval model which predicts relevant premises. HyperTree Proof Search and ABEL (Lample et al., 2022; Gloeckle et al., 2024) introduces an online variant of Monte Carlo Tree Search for the theorem-proving task. PACT (Han et al., 2021) introduces auxiliary training objectives derived from proof state data to learn a better prediction model for search. COPRA (Thakur et al., 2024) uses large LMs as proof step prediction models, which can be conditioned on additional information such as retrieved lemmas, definitions, and execution information, for search. Graph2Tac (Blaauwbroek et al., 2024) learns online hierarchical representations of definitions and theorems, and is used for proof search in Tactician (Blaauwbroek et al., 2020). Several tools have been developed to help with live formalization efforts; these include LLMStep and LeanCopilot for Lean (Welleck & Saha, 2023; Song et al., 2024), and CoqPilot for Coq (Kozyrev et al., 2024).

Previous work has explored providing effective support for measuring models across various interactive theorem provers. `miniF2F` (Zheng et al., 2021) is a multi-language benchmark of high-school competition math problems formalized in Lean 3, HOL Light, Isabelle, and Metamath, though not in Coq. PutnamBench (Tsoukalas et al., 2024) is a collegiate-level benchmark for competition math in Lean 4, Coq, and Isabelle. We do not include evaluations on PutnamBench as our work is not targeted towards olympiad-style theorem-proving. MMA (Jiang et al., 2023) demonstrates that models trained on data from both languages yield downstream performance improvements for autoformalization in both languages, compared to models trained on just one language of data. In our experiments, we demonstrate that such transfer also occurs for neural models trained to perform proof step prediction.

7. Conclusion

We introduced a unified framework for standardized data collection across ITPs like Lean and Coq, which supports proof completion by generating training data, training LMs for proof step prediction, and guiding search algorithms. Using this framework, we produced a multilingual proof step dataset and train the first multi-domain model across multiple ITPs, demonstrating improved transferability between Lean and Coq in mathematics and software verification. Beyond its technical contributions, the framework serves as a foundation for uniting and advancing theorem-proving research communities by providing a shared platform for experimentation and collaboration. In particular, by leveraging this framework, we established that multilingual training not only enables cross-language proof step completion but also outperforms monolingual models, underscoring the benefits of integrating data from diverse formal systems.

In future work, we propose exploring the integration of advanced search algorithms specifically tailored to our standardized framework. This could include developing adaptive search methods that dynamically adjust based on the complexity and characteristics of the theorem being proven. Additionally, further research could focus on optimizing the interaction between the LM and search algorithms to enhance proof efficiency and accuracy. Expanding the dataset to include more diverse ITPs and domains could also improve the model's generalizability and robustness. Finally, investigating the use of reinforcement learning to continuously improve the model based on feedback from successful and failed proof attempts could provide significant advancements in formal theorem proving.

References

Aniva, L., Sun, C., Miranda, B., Barrett, C., and Koyejo, S. Pantograph: A machine-to-machine interaction interface for advanced theorem proving, high level reasoning, and data extraction in lean 4, 2024. URL https://arxiv.org/abs/2410.16429.

Blaauwbroek, L., Urban, J., and Geuvers, H. *The Tactician: A Seamless, Interactive Tactic Learner and Prover for Coq*, pp. 271–277. Springer International Publishing, 2020. ISBN 9783030535186. doi: 10.1007/978-3-030-53518-6_17. URL http://dx.doi.org/10.1007/978-3-030-53518-6_17.

Blaauwbroek, L., Olšák, M., Rute, J., Massolo, F. I. S., Piepenbrock, J., and Pestun, V. Graph2tac: Online representation learning of formal math concepts, 2024. URL https://arxiv.org/abs/2401.02949.

Carrott, P., Saavedra, N., Thompson, K., Lerner, S., Ferreira, J. F., and First, E. Coqpyt: Proof navigation in python in the era of llms. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, volume 21612 of *FSE '24*, pp. 637–641. ACM, July 2024. doi: 10.1145/3663529.3663814. URL http://dx.doi.org/10.1145/3663529.3663814.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019.

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

Gloeckle, F., Limperg, J., Synnaeve, G., and Hayat, A. ABEL: Sample efficient online reinforcement learning for neural theorem proving. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL https://openreview.net/forum?id=kk3mSjVCUO.

Han, J. M., Rute, J., Wu, Y., Ayers, E. W., and Polu, S. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021.

Huet, G., Kahn, G., and Paulin-Mohring, C. The coq proof assistant a tutorial. *Rapport Technique*, 178, 1997.

Jiang, A. Q., Li, W., and Jamnik, M. Multilingual mathematical autoformalization, 2023. URL https://arxiv.org/abs/2311.03755.

Kozyrev, A., Solovev, G., Khramov, N., and Podkopaev, A. Coqpilot, a plugin for llm-based generation of proofs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, pp. 2382–2385. ACM, October 2024. doi: 10.1145/3691620.3695357. URL http://dx.doi.org/10.1145/3691620.3695357.

Lample, G., Lacroix, T., Lachaux, M.-A., Rodriguez, A., Hayat, A., Lavril, T., Ebner, G., and Martinet, X. Hypertree proof search for neural theorem proving. *Advances in Neural Information Processing Systems*, 35:26337–26349, 2022.

Leroy, X. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

Li, Z., Sun, J., Murphy, L., Su, Q., Li, Z., Zhang, X., Yang, K., and Si, X. A survey on deep learning for theorem proving, 2024. URL https://arxiv.org/abs/2404.09939.

Mathcomp. GitHub - math-comp/math-comp: Mathematical Components — github.com. https://github.com/math-comp/math-comp, 2015. [Accessed 01-06-2024].

mathlib Community, T. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20. ACM, January 2020. doi: 10.1145/3372885.3373824. URL http://dx.doi.org/10.1145/3372885.3373824.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pp. 561–577, 2018.

Paulson, L. C. *Isabelle: A generic theorem prover*. Springer, 1994.

Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Sanchez-Stern, A., Alhessi, Y., Saul, L., and Lerner, S. Generating correctness proofs with neural networks. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 1–10, 2020.

Song, P., Yang, K., and Anandkumar, A. Towards large language models as copilots for theorem proving in lean, 2024. URL https://arxiv.org/abs/2404.12534.

Thakur, A., Tsoukalas, G., Wen, Y., Xin, J., and Chaudhuri, S. An in-context learning agent for formal theorem-proving. In *First Conference on Language Modeling*, 2024.

Tsoukalas, G., Lee, J., Jennings, J., Xin, J., Ding, M., Jennings, M., Thakur, A., and Chaudhuri, S. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition, 2024. URL https://arxiv.org/abs/2407.11214.

Wang, Y., Wang, W., Joty, S., and Hoi, S. C. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *EMNLP*, 2021.

Welleck, S. and Saha, R. Llmstep: Llm proofstep suggestions in lean, 2023. URL https://arxiv.org/abs/2310.18457.

Yang, K. and Deng, J. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning*, pp. 6984–6994. PMLR, 2019.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. Leandojo: Theorem proving with retrieval-augmented language models. *arXiv preprint arXiv:2306.15626*, 2023.

Yang, K., Poesia, G., He, J., Li, W., Lauter, K., Chaudhuri, S., and Song, D. Formal mathematical reasoning: A new frontier in ai, 2024. URL https://arxiv.org/abs/2412.16075.

Zheng, K., Han, J. M., and Polu, S. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

Zhu, T., Clune, J., and Welleck, S. Neural theorem proving toolkit, 2023. URL https://github.com/cmu-l3/ntp-toolkit.

A. Appendix

A.1. Training data for proof step prediction modules

The training data is extracted in generic JSON format as shown in Figure 4. We create prompts to train the proof step generation model as demonstrated in Figure 5 from the collected raw data.

B. ITP Versioning and Support

Version Compatibility. Our framework supports multiple versions of Coq (v8.12–v8.18) and Lean 4 (v4.7.0-rc2 and v4.17.0), including those used in MiniF2F and other benchmarks. This flexibility is enabled through standardized JSON-based representations and a language-agnostic prompt format.

Isabelle Integration. We also provide support for Isabelle via the PISA server, which offers a JSON-RPC interface for interaction. However, we did not conduct large-scale training or evaluation with Isabelle due to substantial system-level constraints. Each PISA instance requires a full copy of Isabelle and its heap images (~35 GiB), which scales poorly in parallel settings (e.g., >1 TiB for 50 servers). In addition, memory usage per server can exceed 400 GiB, making distributed parallelism impractical on shared infrastructure. Unlike Lean's lightweight REPL—which we optimized with custom restart logic—PISA is a heavyweight service, and similar fixes are nontrivial. Still, our interface abstractions and data formats already accommodate Isabelle, and future work may enable scalable experiments as tooling improves.

B.1. Parallel Proof Search Beam Algorithm

Figure 6 shows the parallel beam search pseudocode. We utilize the interface module's capabilities to create multiple instances of the proof environment and parallel run tactics to efficiently run search which can be scaled across nodes.

B.2. Hyperparameters used for training PROOFWALA models

We trained our model in a distributed way via PYTORCH and HUGGINGFACE libraries. We used a cluster with 16 Nvidia GH 200 GPU nodes to train PROOFWALA models. The training lasted for approximately 3 days. For further fine-tuning on CategoryTheory data we used the same parameters as shown in Table 4, except we ran only for 1200 steps with a smaller batch size of 8 (the checkpoint step was also accordingly reduced to 1200). Running all our training for various models for a fixed number of steps and the same batch size ensures that every model gets the same number of gradient updates.

(a)

```
{
"theorem_name": "nat_plus_0_is_n"
"start_goals": [
    {
        "hypotheses": [],
        "goal": "forall n : nat, n + 0 = n",
        "# ... extra metadata"
    }
],
"proof_steps": ["intros n."],
"end_goals": [
    {
        "hypotheses": ['n: nat'],
        "goal": "n + 0 = n",
        "# ... extra metadata"
    }
],
"# ... extra metadata"
}
```

(b)

```
{
"theorem_name": "nat_plus_0_is_n"
"start_goals": [
    {
        "hypotheses": [],
        "goal": "∀ (n : Nat), Nat.add n 0
        ↪  = n",
        "# ... extra metadata"
    }
],
"proof_steps": ["intro n"],
"end_goals": [
    {
        "hypotheses": ['n : Nat'],
        "goal": "Nat.add n 0 = n",
        "# ... extra metadata"
    }
],
"# ... extra metadata"
}
```

Figure 4: An excerpt from the extracted training data sequence, $\pi = \langle (O_0, a_1), \ldots, (O_i, a_i), \ldots (O_{n-1}, a_n) \rangle$ (see Section 2), for a given theorem in COQ and LEAN 4. The training data extracted here is used to train PROOFWALA proof step generation models. Here, O_i i.e. set of obligations is extracted under `start_goals` key while O_{i+1} is represented under `end_goals`. The action a_i is extracted as the value of `proof_steps` key. There are more fields other than the ones shown in the figure. (a) Shows an example of a Coq proof step, and (b) shows an example of a Lean proof step.

B.3. Parameters used for proof search

For all our experiments the beam width is 32 (see Table 5), and the temperature for the proof step prediction model is 0.75. We also have a timeout of 600 seconds for each proof attempt for all data mixes except GeoCoq where the timeout was 1200 seconds. Since the proofs in GeoCoq were long (sometimes more than 100 tactics), giving more time for the search to finish was important.

B.4. Bug Fixes in existing framework

Our framework built on top of `coq_serapy`[10] (Sanchez-Stern et al., 2020), while our LEAN 4 implementation is built on top of `REPL`[11] library. We have enhanced these libraries by adding a common abstraction so that data can be collected across multiple languages. We also added ray actors (Moritz et al., 2018) to make it work across clusters on multiple machines. We also fixed some issues with these libraries, for example, `REPL` has a bug that allows it to accept incomplete and incorrect proofs[12]. We also fixed some memory issues which can arise when the REPL library keeps clones of proof-state to allow easy backtracking which leads to exponential memory increase. These fixes were essential for making the framework scalable and run on multiple nodes.

B.5. Proof Tree annotations

Figure 7 shows a visualization generated using our tool. We can use these annotated trees to do qualitative analysis or train models for expert iteration.

C. Examples of Cross-Lingual Transfer in Category Theory

To better understand the cross-lingual transfer discussed in Section 5.1, we examined specific theorems in the Category Theory dataset. We identified Coq theorems that the multilingual model successfully proved, while the Coq-only model failed. For these cases, we found structurally analogous Lean theorems in the training data, particularly those involving categorical notions such as adjunctions, monicity, and currying.

[10] https://github.com/HazardousPeach/coq_serapy
[11] https://github.com/leanprover-community/repl
[12] https://github.com/leanprover-community/repl/issues/44

(a)

```
`Goals to prove:`
`[GOALS]`
`[GOAL] 1`
 S (n + 1) = S (S n)
`[HYPOTHESES] 1`
`[HYPOTHESIS]` IHn : n + 1 = 1 + n
`[HYPOTHESIS]` n : nat
`[END]`
```

(b)

```
`Goals to prove:`
`[GOALS]`
`[GOAL] 1`
 a + x ∈ [a + b-[𝕂]a + c] ↔ x ∈ [b-[𝕂]c]
`[HYPOTHESES] 1`
`[HYPOTHESIS]` 𝕂 : Type u_1
`...`
`[HYPOTHESIS]` π : ι → Type u_6
`[HYPOTHESIS]` inst†⁵ : OrderedRing 𝕂
`...`
`[HYPOTHESIS]` a x b c : E
`[END]`
```

(c)

```
`[RUN TACTIC]`
 auto.
`[END]`
```

(d)

```
`[RUN TACTIC]`
 simp [segment_eq_image']
`[END]`
```

Figure 5: Prompt format for training the proof step generation model. (a) shows the prompt format for COQ , (b) shows the prompt format for LEAN 4, (c) shows the response format used for COQ, and (d) shows the response format used for LEAN 4. We adopted a format similar to the one used in COPRA (Thakur et al., 2024) but without any error context. It is important to note that we do not mention any information about the domain or ITP assistant in the prompt. The prompt format is the same for both languages.

Table 6 shows representative examples. These qualitative results illustrate the model's ability to leverage shared categorical abstractions across ITPs and reinforce our hypothesis about effective structural transfer.

C.1. Qualitative Analysis: Proof Tree Properties

Across various data mixes we observe that proof trees found using the MULTILINGUAL model tend to have more nodes, edges, and higher degrees per node (see Table 7). Figure 8, Figure 9, and Figure 3 show the distribution of nodes, edges, and degrees respectively. Figure 10 shows that MULTILINGUAL often found more proofs for the same theorem during the search.

We observe that MULTILINGUAL model usually searches longer for proofs across the different data mixes. The average time taken to search for proof is summarized in the Table 8 and the distribution of proof search time is shown in Figure 11.

Data Mix	PROOFWALA Model	Avg
LEAN	MULTILINGUAL	2.0363
	LEAN	2.0679
CompCert	MULTILINGUAL	4.7913
	COQ	5.0270
MathComp	MULTILINGUAL	2.4940
	COQ	2.4759
GeoCoq	MULTILINGUAL	9.2486
	COQ	10.6954
CategoryTheory	MULTILINGUAL	3.4909
	COQ	3.0426

Table 9: Summary of *average* proof lengths across various data mixes. The proof lengths are not very different for the two approaches.

Interestingly, we see that there is no significant difference in the size of the proof (number of tactics used) found via the two approaches. Table 9 summarizes the length of the proofs found during the search.

```
PARALLELBEAMSEARCH(O_0, model, t, width)
▷ Set the pool of ITPINTERFACE instances to state O_0
pool ← ITPINTERFACEPOOL.INITIALIZE(O_0)
frontier ← {O_0}
proof_tree ← φ
while frontier ≠ φ
    do
        if TIMEELAPSED(t)
        ▷ Proof not found within the timeout
          then return FALSE, proof_tree
          else 𝕆 ← φ ▷ To store next possible states
               for O ∈ frontier
                   do 𝔸 ← GENERATEPROOFSTEPS(O, model, width)
                      ▷ Filter a sub-pool from ITPINTERFACE instances which are initialized to state O
                      pool' ← pool.FILTER(O)
                      if pool' is empty
                        then pool' ← ITPINTERFACEPOOL.INITIALIZE(O)
                             pool.MERGE(pool') ▷ Merge the new instances to the pool
                      ▷ Execute generated possible proof step(s), 𝔸, in parallel using the pool'
                      𝕆 ← 𝕆 ∧ pool'.EXECUTEPARALLEL(𝔸)
                      ▷ Add all 𝔸 edges in the proof_tree with O as parent
                      if QED ∈ 𝕆
                        then return TRUE, proof_tree
               frontier ← 𝕆
               ▷ Filter the top width states based on some heuristic
               ▷ for example log-likelihood of proof step leading to the state.
               frontier ← frontier.TOPK(width)
return FALSE, proof_tree
```

Figure 6: Pseudocode for the parallel proof search module utilizing Beam Search with the Ray framework (Moritz et al., 2018). This approach enables concurrent exploration of multiple proof steps (tactics) generated by the PROOFWALA model, improving efficiency and throughput. Unlike frameworks such as LeanDojo (Yang et al., 2023) for LEAN 4, which operate sequentially, this module replicates instances of the **interface module** (see Section 3.1) as a custom pool of Ray actors. The custom pool keeps track of ITP instances' proof state. It only uses those instances whose proof state matches the frontier state to continue the exploration (with the occasional overhead of adding more instances to the pool). Each instance in the pool executes potential proof steps in parallel, allowing the search to proceed across various states simultaneously, avoiding the sequential overhead of executing steps one after another on the same ITP instance.

Hyperparameter	Value
Pretrained Model Name	CODET5-BASE (220 M)
Learning Rate	2×10^{-4}
Learning Scheduler Type	`cosine`
Warmup Ratio	`0.03`
Weight Decay	`0.001`
Max Grad Norm (Gradient Clipping)	`0.3`
Optimizer	`adamw_torch`
Gradient Accumulation Steps	`1`
Max # Steps (Gradient Updates)	34000
Batch Size	128
Checkpoint # Steps	20000
Max # Tokens	2048

Table 4: Hyperparameters used for training our PROOFWALA-{LEAN, COQ, MULTILINGUAL}. In line with recent work on training multilingual models for autoformalization (Jiang et al., 2023), we used the same step count and batch sizes to train all our models on different data mixes ensuring our ablation studies about the transfer were fair and were not merely a result of training more on bigger data-mixes.

Parameter	Value
Search Algorithm	Beam Search
Heuristic	Guided by Neg. Log-Likelihood of proof steps predicted by PROOFWALA models
Beam Width	`32`
Timeout	`600` seconds
PROOFWALA model Temp	0.75

Table 5: Parameters used for searching for the complete proof using PROOFWALA models for guidance. We use beam search similar to GPT-f (Polu & Sutskever, 2020).

C.2. Qualitative Analysis: Proofs found by MULTILINGUAL model

Figure 12 shows some of the LEAN 4 and COQ proofs found by MULTILINGUAL model.

Coq Theorem	Lean 4 Equivalent	Multilingual	Coq-Only
`counit_fmap_unit` $\forall\ x,\ \varepsilon \circ \mathsf{fmap}[F]\ \eta \approx \mathsf{id}[F(x)]$	`adjointify_η_ε (X : C)` $F.\mathsf{map}((\mathsf{adjointify}_{\eta,\varepsilon}).\mathsf{hom.app}(X)) \gg \varepsilon.\mathsf{hom.app}(F.\mathsf{obj}(X)) = \nVdash(F.\mathsf{obj}(X))$	✓	✗
`id_monic` $\forall\, x,\ \mathsf{Monic}(\mathsf{id}_x)$	`cancel_mono_id (f : X -> Y)` $g \gg f = f \Leftrightarrow g = \nVdash_X$	✓	✗
`eval_first` $\mathsf{eval} \circ \mathsf{first}(f) \approx \mathsf{uncurry}(f)$	`uncurry_id_eq_ev (A X : C)` $\mathsf{uncurry}(\nVdash_{A \Rightarrow X}) = (\mathsf{exp.ev}(A)).\mathsf{app}(X)$	✓	✗

Table 6: Examples of Category Theory theorems in Coq and their Lean equivalents. The multilingual model succeeds, while the Coq-only model fails.

Data-Mix	PROOFWALA Model	Avg. Proof Tree Stats		
		# Nodes	# Edges	# Degree
LEAN	LEAN	3.989	3.536	1.536
	MULTILINGUAL	4.729	4.689	1.983
MathComp	COQ	2.534	1.739	1.167
	MULTILINGUAL	2.576	1.822	1.207
GeoCoq	COQ	15.358	14.457	1.180
	MULTILINGUAL	17.144	15.75	1.353
CompCert	COQ	8.048	7.480	1.404
	MULTILINGUAL	8.318	8.200	1.584
CategoryTheory	COQ-CAT-THEORY	5.674	5.804	2.301
	MULTILINGUAL-CAT-THEORY	7.056	7.130	2.193

Table 7: Comparison between the average number of nodes, edges, and degree of the proof trees generated on various PROOFWALA models over different data-mixes.

Data Mix	PROOFWALA Model	Avg
LEAN	MULTILINGUAL	30.5296
	LEAN	20.7370
CompCert	MULTILINGUAL	59.9424
	COQ	70.3282
MathComp	MULTILINGUAL	8.9993
	COQ	7.5634
GeoCoq	MULTILINGUAL	107.1999
	COQ	74.3914
CategoryTheory	MULTILINGUAL	39.2182
	COQ	27.4105

Table 8: Summary of *average* proof times (in seconds) across various data mixes. We can see that PROOFWALA-MULTILINGUAL usually searches for longer and hence the average time is higher. Since the proof trees generated are larger for MULTILINGUAL approach, it is reasonable that overall proof search time will be higher.

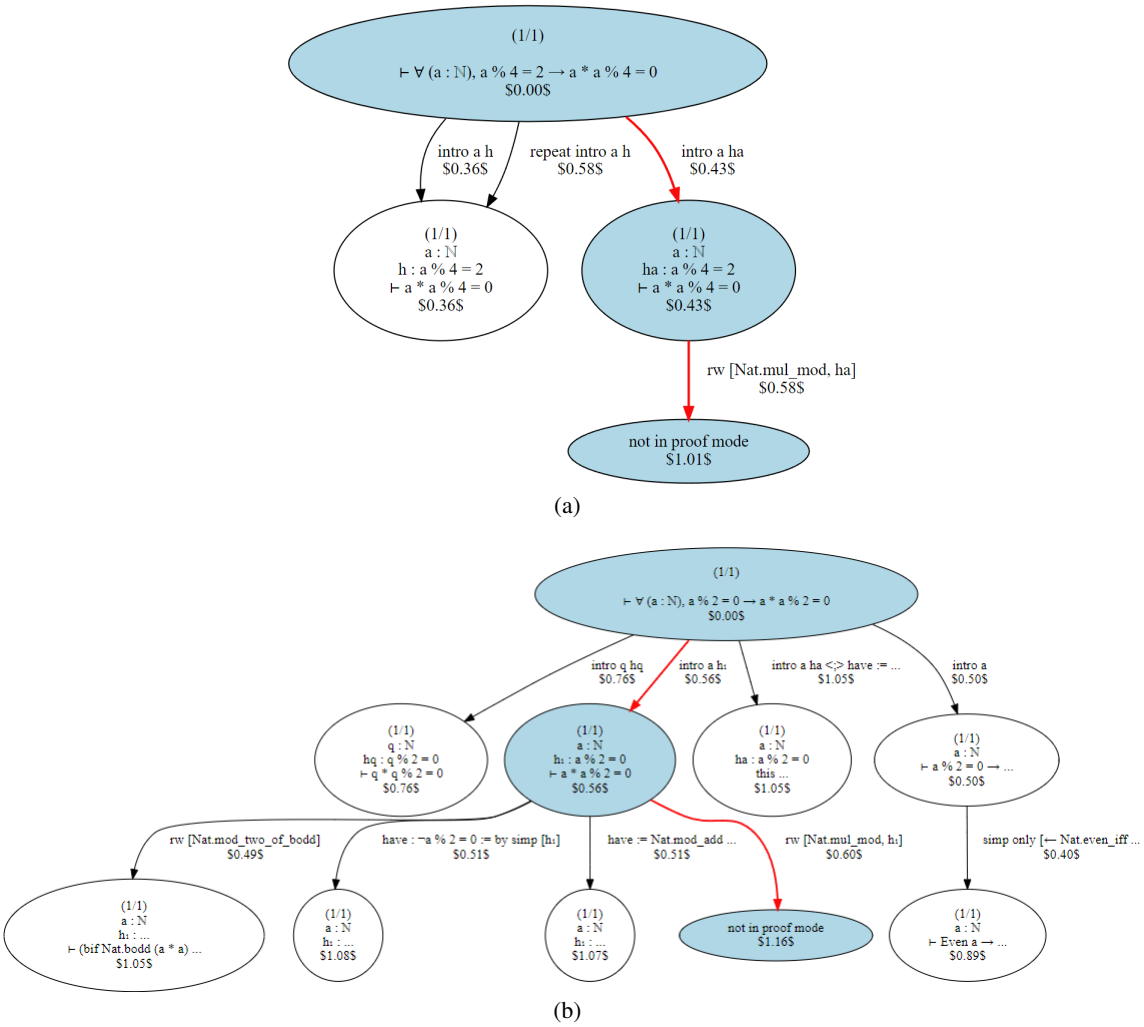

Figure 7: Visualization of the proof trees generated via the **Proof Search Module** (see Section 3.3) for Lean 4 theorems stating: (a) $\forall(a : \mathbb{N}), a \,\%\, 4 = 2 \rightarrow a * a \,\%\, 4 = 0$, and (b) $\forall(a : \mathbb{N}), a \,\%\, 2 = 0 \rightarrow a * a \,\%\, 2 = 0$. The proof tree can be annotated with the correct proof path, and scores for each edge (proof step) and node (proof-state). This tree has been generated through Beam Search guided by the PROOFWALA-MULTILINGUAL model, the framework also supports best first search. The tree only includes proof steps (edges) that can be applied to the given proof-state (node) without any error. The numbers within the $ symbols are the negative log-likelihood of the tokens generated by the PROOFWALA-MULTILINGUAL proof step generation model.

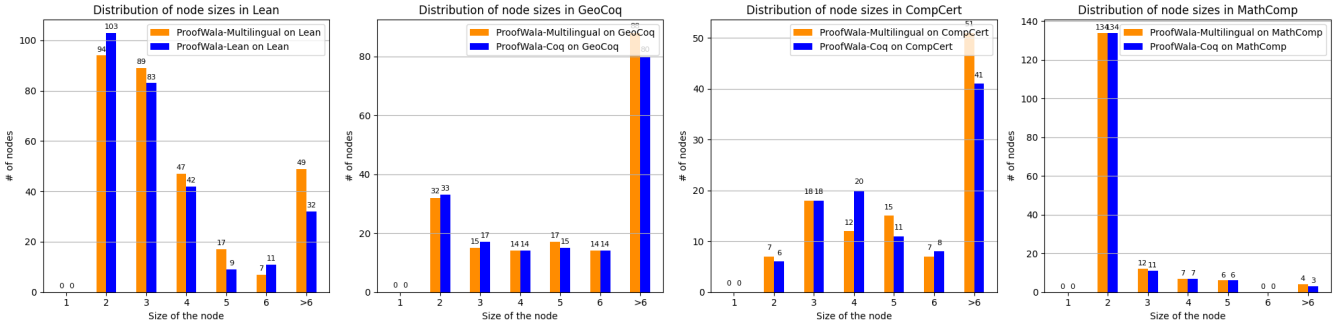

Figure 8: Distribution of proof-tree nodes across various data-mixes found by different PROOFWALA models. It is interesting to note that across all data-mixes, the PROOFWALA-MULTILINGUAL model tends to produce more nodes per proof tree. This indicates that PROOFWALA-MULTILINGUAL often constructs larger proof trees during search.

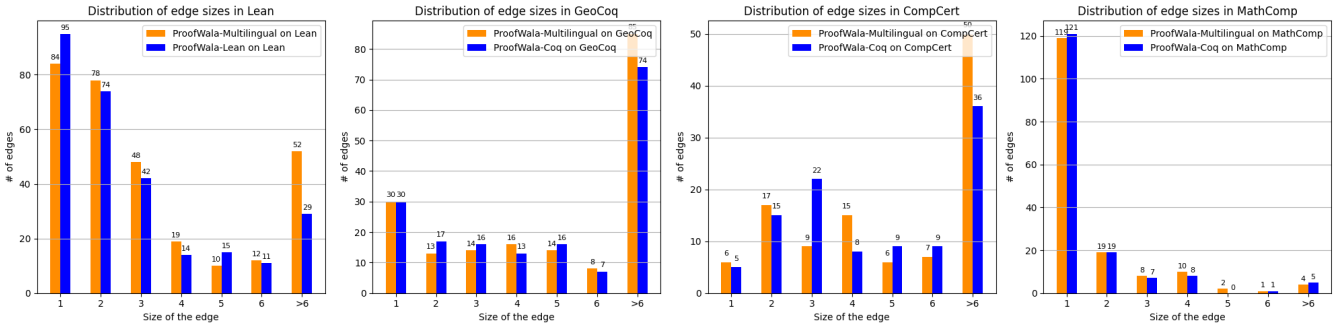

Figure 9: Distribution of proof-tree edges across various data-mixes found by different PROOFWALA models. It is interesting to note that across all data mixes, PROOFWALA-MULTILINGUAL models tend to have more edges per proof tree. This indicates that PROOFWALA-MULTILINGUAL often find more compilable tactics while searching to complete the proof.

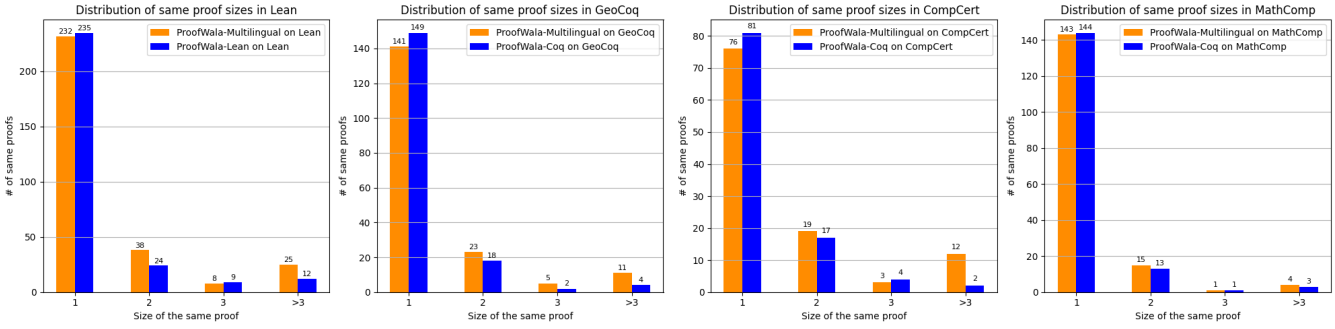

Figure 10: Distribution of the number of proofs found for the same theorem across various data-mixes found by different PROOFWALA models. It is interesting to note that across all data mixes, the PROOFWALA-MULTILINGUAL model tends to produce more proofs for the same theorem.

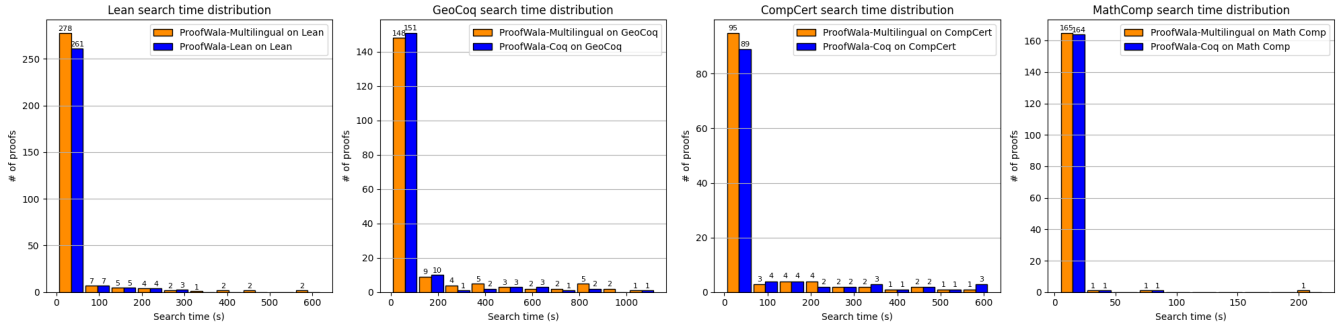

Figure 11: Distribution of the time taken to find proofs across various data-mixes found by different PROOFWALA models. We can see that across all data mixes, the PROOFWALA-MULTILINGUAL model tend to run longer proof searches, and thus effectively search more.

(a)

```
theorem lapMatrix_toLin'_apply_eq_zero_iff_forall_reachable (x : V → ℝ) :
    Matrix.toLin' (G.lapMatrix ℝ) x = 0 ↔ ∀ i j : V, G.Reachable i j → x i = x j  := by
    rw ←[ (posSemidef_lapMatrix ℝ G).toLinearMap₂'_zero_iff, star_trivial,
        lapMatrix_toLinearMap₂'_apply'_eq_zero_iff_forall_adj]
    refine ⟨?_, fun h i j hA ↦ h i j hA.reachable⟩
    intro h i j ⟨w⟩
    induction' w with w i j _ hA _ h'
    rfl
    exact (h i j hA).trans h'
```

(b)

```
theorem interval_average_symm (f : ℝ → E) (a b : ℝ) : ∫ x in a..b, f x) = ∫ x in b..a, f
     x  := by
    simp only [intervalIntegral, setAverage_eq, smul_sub]
    obtain rfl | hab := eq_or_ne a b
    rfl
    rw [uIoc_comm a b, uIoc_comm b a]
```

(c)

```
Theorem coplanar_perm_11  : forall A B C D,
  Coplanar A B C D -> Coplanar B D C A.
Proof.
    intros A B C D HCop.
    destruct HCop as [X H]; exists X.
    induction H; try (induction H); spliter; Col5.
Qed.
```

(d)

```
Theorem coprimepP: forall  p q ,
 reflect (forall d, d %| p -> d %| q -> d %= 1) (coprimep p q).
Proof.
    rewrite /coprimep; apply: (iffP idP) => [/eqP hs d dvddp dvddq | h].
    have/dvdp_eqp1: d %| gcdp p q by rewrite dvdp_gcd dvddp dvddq.
    by rewrite -size_poly_eq1 hs; exact.
    by rewrite size_poly_eq1; case/andP: (dvdp_gcdlr p q); apply: h.
Qed.
```

Figure 12: Some proofs discovered by PROOFWALA-MULTILINGUAL in our experiments on theorems from Mathlib, GeoCoq, and MathComp.