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

Amitayush Thakur
University of Texas, Austin, TX, USA

George Tsoukalas
University of Texas, Austin, TX, USA

Greg Durrett
University of Texas, Austin, TX, USA

Swarat Chaudhuri
University of Texas, Austin, TX, USA

Abstract

Neural approaches to theorem proving require robust infrastructure for interfacing with interactive theorem provers (ITPs), extracting structured proof data, and executing proof search at scale. However, existing tooling is often assistant-specific and oriented toward interactive, file-level execution, making repository-scale analysis and parallel experimentation challenging. We present ProofWala, a multilingual proof engineering framework built around `itp-interface`, a reusable library for programmatic interaction with ITPs. For Lean 4, we implement a meta-programmed interaction layer that executes inside the elaborator, enabling semantically faithful tactic-level tracing together with declaration- and dependency-level extraction across entire repositories. This design extends beyond traditional REPL-style interaction by supporting project-wide analysis, environment cloning, and pooled execution of proof states. The implementation is robust across Lean 4 versions after 4.15.0 with forward compatibility support. The same interface abstraction supports tactic execution and data extraction for multiple versions of Rocq, yielding a unified cross-assistant pipeline.

Built on this infrastructure, ProofWala provides standardized multilingual proof datasets, model training utilities, and parallel proof search algorithms. Using the framework, we demonstrate that multilingual training across Lean and Rocq enables cross-lingual and cross-domain transfer. In particular, we observe statistically significant improvements on the largest benchmark (Lean/Mathlib) and in domain adaptation (CategoryTheory), while other settings exhibit consistent but not statistically significant trends in favor of multilingual training. We open-source the full framework, including the parallel proof search module, the `itp-interface` library, multilingual datasets, and trained models, providing a scalable foundation for proof mining, neural theorem proving, and cross-assistant experimentation.

2012 ACM Subject Classification Theory of computation → Logic and verification; Software and its engineering → Formal software verification; Computing methodologies → Machine learning

Keywords and phrases neural theorem proving, ITP, automated reasoning, LLM guided theorem proving, ITP framework

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.24

Supplementary Material
Software (Source Code): `https://github.com/trishullab/itp-interface`
archived at `swh:1:dir:b3d62a8032c04e35c781d786f4e49cc20110b016`
Software (Source Code): `https://github.com/trishullab/proof-wala`
archived at `swh:1:dir:eadfe02d90575aaf2fbaed5c3c97bcdb6daee16e`

Declaration about GenAI/LLM use Large language models were used to assist with language editing, proofreading, and improving the presentation of the manuscript. All technical content, experimental design, implementation, and scientific conclusions were developed and verified by the authors.

© Amitayush Thakur, George Tsoukalas, Greg Durrett, and Swarat Chaudhuri;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Automated theorem proving has long been considered a grand challenge in artificial intelligence. Recently, deep learning has emerged as a promising approach to this challenge [14, 27]. Broadly, neural methods for theorem proving use language models to generate proofs in the language of an *interactive theorem prover* (ITP), e.g., Lean [6], Rocq [9], or Isabelle [18]. An ITP represents proofs as sequences of simplification steps, or *tactics*, and mechanically checks such proofs for correctness. From this perspective, automated theorem proving amounts to generating a tactic sequence that passes the ITP's checks.

Most neural approaches follow the strategy proposed by Polu et al. [19]: train a generative language model (LM) to predict tactics (and their parameters) conditioned on the current proof state, using proof-step data extracted from existing formal repositories, and then wrap the model with a search procedure that conducts proof search (see Section 2).

While neural theorem proving is gaining momentum, the supporting *tooling infrastructure* remains fragmented. Existing pipelines for data extraction and proof interaction are often ITP-specific, rely on brittle, assistant-dependent representations, and provide limited support for scalable execution (e.g., parallel tactic evaluation, cloning proof states, or traversing large dependency graphs). This fragmentation hinders systematic comparisons, slows down proof-engineering workflows, and makes it difficult to exploit potential cross-lingual and cross-domain gains from training on multilingual formal corpora.

In response, we introduce ProofWala[1], a multilingual framework for dataset collection, interaction, training, and proof search across ITPs and domains. A central component of ProofWala is `itp-interface`, a reusable library for programmatic interaction with proof assistants. For Lean, we deliberately avoid a Lean REPL[2]-based architecture. While REPL-style interfaces are well suited for interactive, file-by-file tactic execution, they primarily expose sequential tactic traces and provide limited access to declaration metadata or project-wide structure. In particular, extracting declaration-level information (e.g., inductive definitions, constants, namespaces, and imports) and constructing repository-wide dependency graphs is difficult or infeasible through a REPL workflow. Such capabilities are essential for our goals of dependency-aware data extraction, large-scale repository mining, and cloning proof states for parallel execution.

Instead, we implement Lean meta-programs that run directly inside the elaborator, enabling semantically faithful tactic-level tracing together with declaration and dependency extraction across an entire project. This design supports environment cloning and pooling, which in turn enables parallel proof search and high-throughput data collection. The resulting implementation is robust across Lean releases starting from 4.15.0 via a thin compatibility layer that isolates internal API changes and provides largely automatic forward compatibility. The same interface abstraction supports tactic execution and data extraction for Rocq (multiple versions), yielding a unified cross-assistant pipeline.

Compared to existing Lean tooling such as Pantograph[3][1], which focuses primarily on interactive orchestration of a single Lean instance, `itp-interface` is designed as a scalable infrastructure layer: it supports repository-level extraction, declaration graph construction, multi-environment execution pools, and distributed parallelism (e.g., via Ray [17], a framework for parallel and distributed execution across processes or machines), as well as multilingual

[1] "Wala" is a suffix from Indic languages (often spelled "wallah"), meaning "one who is associated with or provides a particular thing."
[2] `https://github.com/leanprover-community/repl`
[3] `https://github.com/leanprover/Pantograph`

(Lean and Rocq) interoperability. These capabilities are central to large-scale proof mining and parallel proof search, and extend beyond typical REPL- or single-environment-based workflows.

Beyond interaction and extraction, ProofWala integrates efficient proof search algorithms, including parallelized best-first and beam search, enabled by an *environment pool* abstraction that maintains multiple proof environments initialized to identical frontier states. This pool allows executing candidate tactics concurrently, and integrates naturally with Ray-based distributed execution for parallel proof search and large-scale annotation.

We release an open-source code library, multilingual datasets, and fine-tuned models that facilitate end-to-end proof search in Lean 4 and Rocq. Using ProofWala, we demonstrate that multilingual training can yield positive cross-lingual and cross-domain transfer, improving proof step generation across different assistants and domains.

In summary, our work makes three key contributions:

A Standardized Framework and Engineering Library. We propose ProofWala, built around `itp-interface`, a unified framework for extracting and organizing tactic-level training data and supporting tool development for formal theorem proving in Lean and Rocq. For Lean, we introduce a meta-programmed instrumentation pipeline that supports (i) tactic-level execution with semantically faithful proof-state tracing, including improved handling of nested `have` blocks in tactic mode, and (ii) dependency-aware repository traversal for extracting declarations and their dependency relations across entire Lean projects. Our Lean backend is version-robust across Lean 4 releases 4.15.0+ (largely automatic forward compatibility). For Rocq, we support tactic execution and data extraction across multiple versions, enabling a consistent unified data format and prompt-formatting scheme across ITPs and domains. All code is open source: the ProofWala framework can be found here, and `itp-interface` can be found here. We additionally provide a Streamlit [22]-based *Lean Declaration Database Explorer* – an interactive web application for data exploration that enables querying extracted SQLite databases, searching declarations, and visualizing dependency structure. All our fine-tuned models ProofWala-{Lean, Rocq, and Multilingual} on HuggingFace for public use (URL: Lean, Rocq and Multilingual).

Support for Parallel Proof Completion. Similar to *GPT-f* [19], the framework supports proof completion via search guided by a proof step generation model. We make proof search ITP-agnostic and parallel by introducing an *environment pool* that clones proof environments and executes tactics in parallel across these replicas (scaling across machines using Ray). This enables high-throughput exploration, proof-tree annotation, and visualization. To the best of our knowledge, ProofWala is the first open-source framework to provide parallel proof search by explicitly supporting cloned proof environments and concurrent tactic evaluation within a unified multi-ITP interface.

Demonstration of Cross-Lingual and Cross-Domain Transfer. Using ProofWala, we study the effect of multilingual proof data on proof step prediction, and observe evidence of cross-domain and cross-lingual transfer for both Lean and Rocq, across general mathematics and software verification. We observe statistically significant improvements on the largest benchmark (Lean/Mathlib) and in domain adaptation (CategoryTheory), while other settings exhibit consistent but not statistically significant trends in favor of multilingual training.

We release the multi-domain multilingual training data (about 450K datapoints; 270M tokens; about 80K theorems) and multiple open models trained on different data mixtures. Our ProofWala-Multilingual model is an open proof step generation model trained on diverse domains and ITPs, and can be directly used for proof search in both formal mathematics and software verification.

2 Problem Formulation

We model interactive theorem proving as a structured state-transition system operating over *proof obligations*. Each obligation o is a pair (g, h), where g is the current goal and h is the associated local context (hypotheses). A proof state O is a finite set of such obligations. The objective of the prover is to transform an initial state into a terminal state in which all obligations have been discharged.

Formally, we abstract an ITP as a *proof environment* consisting of:

- A set of states $\mathcal{O}$, where each state $O = \{o_1, \ldots, o_k\}$ is a set of obligations.
- An initial state $\mathcal{I} = \{(g_{in}, h_{in})\}$ extracted from a user-provided theorem.
- A distinguished goal state `QED`, corresponding to the empty obligation set.
- A finite (or effectively enumerable) set of tactics.
- A transition function $T(O, a)$ describing the result of applying tactic a to state O.

If tactic a is successfully applied at state O, then $T(O, a)$ yields a new set of obligations reflecting the updated proof state. If a is invalid or fails, we treat the transition as leaving the state unchanged.[4]

We extend T to sequences of tactics $\alpha = \langle a_1, \ldots, a_n \rangle$ via:

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

▶ **Problem 1** (Theorem Proving). *Given an initial state O_{in}, find a tactic sequence α such that* $T_{seq}(O_{in}, \alpha) = \texttt{QED}$.

This abstraction is independent of any specific proof assistant and underlies the design of `itp-interface`. In our implementation, Lean 4 and Coq proof states are mapped into this unified representation via meta-programmed instrumentation (Lean) and assistant-specific interaction layers (Coq). This uniform state model enables assistant-agnostic data extraction, dependency-aware analysis, environment cloning, and parallel proof search.

Figure 1 shows a LEANtheorem and proof discovered using our framework. The proof proceeds by successively transforming the initial obligation through tactic applications until the obligation set is empty.

Model-Guided Search. To automate the search for α, we employ a model that estimates the likelihood of applying tactic a at state O, denoted $p(a \mid O)$. In practice, this model may be implemented as a language model that generates tokenized tactic representations conditioned on a serialized form of O. We omit tokenization details for clarity. Because our state abstraction is standardized across assistants, models can be trained on multilingual proof data and applied uniformly across Lean and Rocq environments.

[4] In practice, failure is captured explicitly by the underlying ITP; our abstraction omits low-level error states for clarity.

```
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`.

The search module operates over the state space defined above, using $p(a \mid O)$ to guide node expansion. Crucially, our environment-pooling abstraction allows multiple frontier states to be evaluated in parallel, enabling scalable exploration of the proof tree across cloned proof environments.

3 The ProofWala Framework

We now describe the ProofWala framework. Our primary motivation is to provide a reusable, language-agnostic infrastructure for interacting with interactive theorem provers (ITPs), extracting structured proof data at scale, and supporting programmable proof search. Rather than focusing solely on model training, we design ProofWala as a systems layer that enables repository-wide mining, dependency-aware analysis, environment cloning, and scalable experimentation across assistants.

The framework is organized around three components:

1. **Interface Module** (`itp-interface`): a reusable library for programmatic interaction with ITPs.
2. **Data and Model Module**: infrastructure for extracting proof traces and training proof-step predictors.
3. **Parallel Proof Search Module**: scalable exploration of proof states using environment pooling.

Figure 2 illustrates how these components interact.

3.1 Interface Module

The `itp-interface` library provides a unified abstraction for interacting with multiple ITPs. In particular, the interface module supports interaction with Lean 4 and Rocq (multiple versions from 10.0 to 18.0), and exposes a standardized state-transition API consistent with the formulation in Section 2. Despite differences in internal architecture, both assistants are mapped into a common representation of proof states and tactic transitions.

Lean 4 Backend via Meta-Programming. Our Lean backend is implemented using Lean 4 meta-programming rather than relying solely on external orchestration. The interaction layer executes inside the elaborator, enabling:

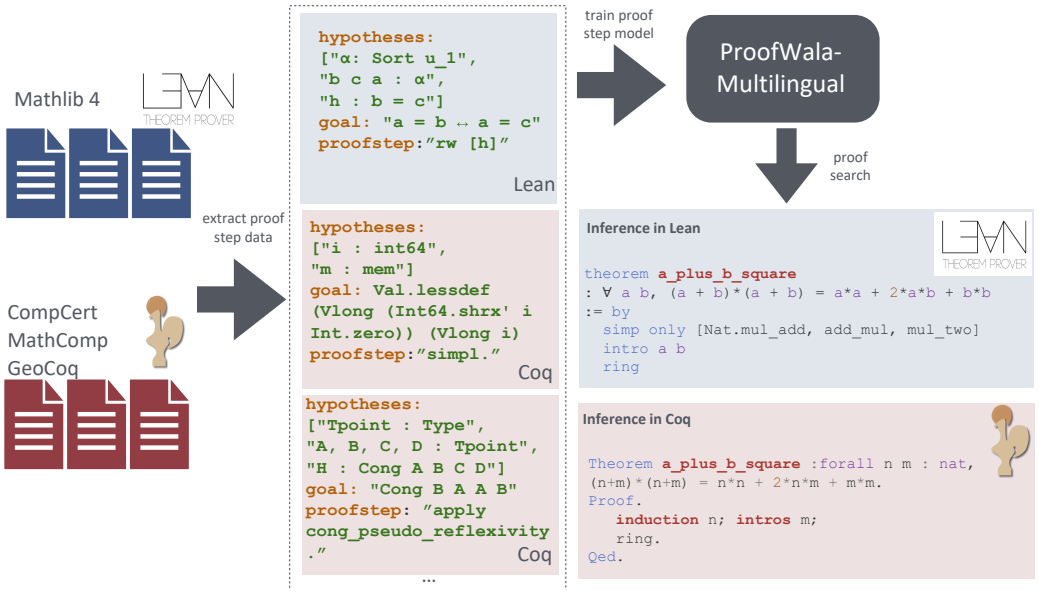

Figure 2 Overview of the ProofWala framework. The interface module enables repository-scale extraction and programmable interaction with Lean and Rocq. Extracted traces are used for training proof-step predictors. The search module executes tactics – potentially in parallel – using cloned proof environments.

- semantically faithful tactic-level execution and state tracing,
- extraction of declaration-level metadata (e.g., inductive definitions, constants, namespaces),
- construction of repository-wide dependency graphs,
- improved handling of nested tactic constructs (e.g., `have`, `by`, and subgoal generation),
- cloning of proof environments for scalable parallel execution.

This design enables repository-scale mining and dependency-aware extraction that extends beyond file-scoped interaction workflows. The implementation is compatible with Lean 4 versions 4.15.0 through 4.24.0 and employs a thin compatibility layer to isolate internal API changes, providing largely automatic forward support for future versions.

Rocq Backend. For Rocq, we build on top of `coq_serapy`[5] [20], which provides structured access to Rocq's proof engine. Our backend extends this infrastructure to support systematic extraction of proof states and tactic traces. The system supports Rocq (formerly Coq) versions 8.10 through 8.18 and maps assistant-specific state representations into the same unified format used for Lean. Although the internal interaction mechanism differs from the Lean meta-programmed backend, both assistants expose a consistent abstract interface to the rest of the framework.

Environment Pooling and Parallel Execution. A key feature of `itp-interface` is the ability to clone proof environments and maintain pools of independent instances initialized to identical states. This abstraction enables concurrent tactic execution and forms the

[5] `https://github.com/HazardousPeach/coq_serapy`

foundation of our parallel proof search module. The pool can be integrated with distributed execution frameworks such as Ray, enabling scalable proof search and large-scale proof annotation in both Lean and Rocq.

Declaration Database and Tooling. Beyond tactic tracing, the interface layer extracts declaration metadata and dependency edges into a structured SQLite database. We provide a Streamlit-based *Lean Declaration Database Explorer* for interactive querying, dependency visualization, and connected-component analysis of large formal repositories. This tooling supports repository-level inspection, empirical studies of formal developments, and proof engineering workflows.

Generality and Limitations Across ITPs. While the interface abstraction is designed to be assistant-agnostic, our large-scale experiments focus on Lean and Rocq, which share a tactic-based interaction model. Extending this abstraction to ITPs with fundamentally different interaction paradigms introduces additional challenges. For example, Isabelle follows a document-oriented architecture in which proofs are processed as structured theory files rather than through stepwise tactic execution.

We implemented preliminary support for Isabelle via the PISA server, which exposes a JSON-RPC interface compatible with our framework. However, in practice, this integration incurs substantial system-level overhead: each PISA instance requires a full Isabelle installation and heap image (tens of gigabytes), leading to prohibitive memory and storage requirements when scaling to many parallel environments. In contrast to the lightweight environment cloning used for Lean and Rocq, this makes large-scale parallel proof search impractical with current tooling.

Despite these limitations, our core abstractions – standardized state representations, tactic traces, and data formats – remain compatible with such systems. We therefore view full support for document-oriented assistants like Isabelle as an engineering challenge rather than a conceptual limitation, and leave scalable integration as an important direction for future work.

3.2 Parallel Proof Search Module

The proof search module operates over the state-transition system defined in Section 2. Given a proof state O, the search procedure generates candidate tactics and applies them to produce successor states. We implement beam search and best-first search strategies over this state space.

Environment Pooling and Parallel Execution. A central contribution of this module is its integration with the environment pool abstraction introduced in Section 3.1. Instead of executing candidate tactics sequentially within a single ITP instance, we dispatch tactic applications across multiple cloned proof environments. Each environment instance maintains an independent proof state. The pool dynamically filters instances to match frontier states during search and creates additional instances when needed.

This design enables concurrent exploration of multiple branches of the proof tree. When integrated with Ray [17], the pool can distribute tactic execution across processes or machines, allowing scalable proof search in both Lean and Rocq.

Parallel Beam Search. Figure 4 presents pseudocode for our parallel beam search implementation. The search maintains a frontier of states, generates candidate tactics using a guidance model, and executes these tactics in parallel across matching environment instances. The pool ensures that only environments whose internal state matches the current frontier state are reused, avoiding unnecessary reinitialization overhead.

Compared to sequential search frameworks (e.g., PANTOGRAPH [1], LeanDojo [28] for LEAN), our design avoids repeatedly executing tactics in a single process and instead explores multiple branches of the proof tree simultaneously. This improves throughput and enables scalable experimentation often needed for more challenging theorems.

Proof Tree Annotation and Visualization. During search, we maintain an annotated proof tree recording states, tactic applications, and transition scores. Each node corresponds to a proof state, and each edge corresponds to a successfully executed tactic. Because tactics are only added when they execute without error in the underlying ITP, the tree represents valid state transitions in the proof environment.

These annotated trees serve multiple purposes: (1) qualitative analysis of search behavior, (2) visualization of explored proof branches, (3) debugging and proof engineering, and (4) potential training signals for iterative refinement methods.

Figure 3 shows example proof trees generated during beam search for LEAN theorems. Edge labels indicate tactic scores (e.g., negative log-likelihood under a guidance model), and the correct proof path can be highlighted within the explored tree.

4 Dataset & Model Details

This section describes the datasets constructed using the PROOFWALA extraction pipeline and the training configurations used to validate the framework. The goal is to document the scale, diversity, and structure of the extracted datasets, as well as the model configurations built on top of them. Experimental results are presented separately in Section 5.1.

4.1 Dataset Construction

Using the `itp-interface` module, we extract proof-state–tactic pairs from mature repositories in both LEAN and ROCQ. The extraction process replays tactic scripts inside the assistant's kernel and records semantically valid state transitions in the unified representation described in Section 2. This ensures that all extracted data corresponds to valid kernel-level proof execution.

We extract data from the following repositories: (1) **CompCert** [6], (2) **MathComp** [7], (3) **GeoCoq** [8], (4) **Mathlib4** [9], and (5) **CategoryTheory** [10]

For CompCert and Mathlib, we reuse established train/test splits from prior work [20, 28]. For the remaining repositories, we construct theorem-level splits, ensuring at least 500 test theorems when possible.

[6] `https://github.com/AbsInt/CompCert`
[7] `https://github.com/math-comp/math-comp`
[8] `https://github.com/GeoCoq/GeoCoq`
[9] `https://github.com/leanprover-community/mathlib4`
[10] `https://github.com/jwiegley/category-theory`

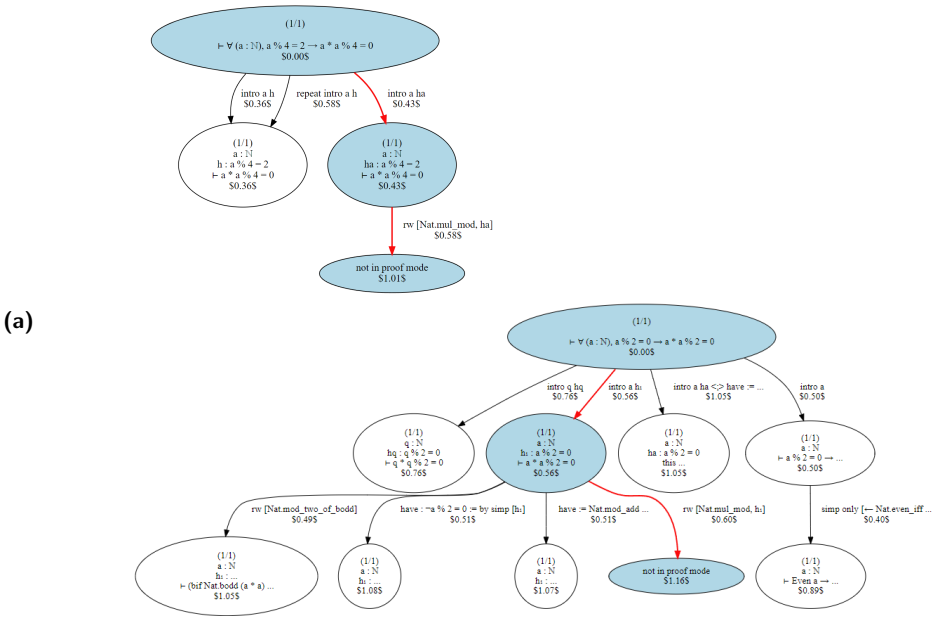

Figure 3 Visualization of proof trees generated by the parallel proof search module for Lean theorems: (a) $\forall (a : \mathbb{N}), a \,\%\, 4 = 2 \to a * a \,\%\, 4 = 0$, (b) $\forall (a : \mathbb{N}), a \,\%\, 2 = 0 \to a * a \,\%\, 2 = 0$. Nodes represent proof states; edges correspond to successfully executed tactics. Edge labels (within '$') denote tactic scores.

```
ParallelBeamSearch(O_0, model, t, width)
 1  ▷ Initialize pool with environments at initial state
 2  pool ← ItpInterfacePool.Initialize(O_0)
 3  frontier ← {O_0}
 4  proof_tree ← ∅
 5  while frontier ≠ ∅
 6        do
 7           if TimeElapsed(t)
 8              then return False, proof_tree
 9           𝕆 ← ∅ ▷ Next frontier states
10           for O ∈ frontier
11                 do
12                    𝔸 ← GenerateProofSteps(O, model, width)
13                    ▷ Reuse environments already at state O
14                    pool′ ← pool.Filter(O)
15                    if pool′ is empty
16                       then
17                             pool′ ← ItpInterfacePool.Initialize(O)
18                             ▷ Register new environments
19                             pool.Merge(pool′)
20                    ▷ Execute candidate tactics in parallel
21                    (𝕆′, pool′) ← pool′.ExecuteParallel(𝔸)
22                    pool.UpdateState(pool′) ▷ Re-index new states
23                    𝕆 ← 𝕆 ∪ 𝕆′
24                    if QED ∈ 𝕆
25                       then return True, proof_tree
26           ▷ Beam pruning
27           frontier ← 𝕆.TopK(width)
28  return False, proof_tree
```

Figure 4 Parallel beam search using an environment pool. Candidate tactics are executed concurrently across cloned ITP instances. The pool maintains a mapping from proof states to environment instances and is updated after each execution step.

Table 1 Overview of multilingual datasets constructed and used for training and evaluation.

(a) Data-mixes constructed using the unified ProofWala extraction pipeline.

Data-mix	Source	Model Trained	Token Count
Initial Fine-tuning			
1. CompCert	CompCert Repo	–	61.6M
2. MathComp	MathComp Repo	–	18.2M
3. GeoCoq	GeoCoq Repo	–	91.2M
4. Rocq	Mix 1–3	Rocq	171M
5. Lean	Mathlib Repo	Lean	99M
6. Multilingual	Mix 4–5	Multilingual	270M
Further Fine-tuning			
7. CategoryTheory	CategoryTheory Repo	Multilingual-Cat-Theory & Rocq-Cat-Theory	1.7M

(b) Dataset sizes under the unified ProofWala representation.

	# 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. Rocq	205604	20072	7213	20857	1542	937
5. Lean	237003	4323	4220	56140	991	1035
6. Multilingual	442607	24395	11433	76997	2533	1972
7. CategoryTheory	4114	610	208	573	101	43

Across all repositories, we extract `442,607` proof-step/state pairs from `76,997` theorems spanning both Lean and Rocq. The CategoryTheory dataset is excluded from initial training and reserved for domain adaptation experiments.

To evaluate different training configurations, we construct several data-mixes combining monolingual and multilingual datasets. These are summarized in Table 1a.

Table 1b reports the number of extracted proof-step/state pairs and theorem counts for each data-mix. These statistics illustrate the scale and diversity of the extracted multilingual dataset.

4.2 Training Data Representation for Proof Step Prediction

The extracted proof traces are stored in a unified `JSON` format that abstracts over assistant-specific details while preserving full semantic information about proof states and tactic transitions. Each training example corresponds to a transition (O_i, a_i, O_{i+1}) as defined in Section 2, where: (1) O_i (start_goals) represents the current set of proof obligations, (2) a_i (proof_steps) represents the applied tactic, and (3) O_{i+1} (end_goals) represents the resulting obligations.

An excerpt of the extracted data format for Rocq and Lean 4 is shown in Figure 5. Although the underlying assistants differ in syntax and internal representations, the stored format is structurally identical across languages.

(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 5 Excerpt from the extracted proof-step data in unified JSON format. Each entry represents a state transition (O_i, a_i, O_{i+1}). The format is identical across Rocq and Lean 4. Additional metadata fields (not shown) include file paths, namespaces, and dependency information.

From this unified JSON representation, we construct standardized prompts for proof step prediction. The prompt encodes the current proof state (goals and hypotheses) and requests the next tactic. Importantly, the prompt format is identical across Lean and Rocq, and does not reveal the originating assistant or repository.

This design enables cross-assistant training and evaluation without introducing assistant-specific tokens into the model's input.

4.3 Model Configuration

To validate the usability of the extracted datasets, we fine-tune a pretrained CodeT5-Base model (220M parameters) for proof step prediction.

We train three configurations: (1) ProofWala-Rocq (trained on Rocq repositories), (2) ProofWala-Lean (trained on Lean repositories), and (3) ProofWala-Multilingual (trained jointly on both assistants).

All models are trained for the same number of optimization steps and batch sizes to ensure comparability across configurations. Hyperparameters are provided in Table 2.

These trained models are used in the parallel proof search module described in Section 3.2. Experimental results evaluating these models are presented in Section 5.1.

Table 2 Hyperparameters used for training our ProofWala-{Lean, Rocq, Multilingual}. In line with recent work on training multilingual models for autoformalization [10], 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.

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

Model choice and scope of evaluation. All experiments in this work use CodeT5-Base (220M parameters) as the underlying proof-step prediction model. This choice is driven by our experimental goals. First, we train multiple models across different data-mixes (monolingual and multilingual) to study transfer effects under controlled conditions; using a moderate-sized model allows us to keep training budgets comparable across these settings. Second, our primary objective is to evaluate the capabilities of the ProofWala framework – including unified data extraction, training, and parallel proof search – rather than to optimize absolute theorem-proving performance. Since the framework is model-agnostic and built on top of standard HuggingFace training pipelines, its functionality and scalability are largely independent of the specific model size.

While larger models and more complex systems (e.g., with retrieval or verifier augmentation) achieve stronger results on benchmark datasets, we intentionally use a fixed-capacity model to isolate the effect of multilingual training on proof-step prediction and search. The improvements observed for the multilingual model over monolingual baselines therefore reflect data-driven transfer rather than gains from increased model capacity. We expect these transfer effects to persist – and potentially strengthen – when scaling to larger models, although verifying this empirically remains an important direction for future work.

5 Evaluation

Using our trained ProofWala models, we evaluate (i) the benefit of incorporating multilingual proof data into proof-step prediction and end-to-end proof search, and (ii) whether multilingual pretraining improves adaptation to a novel domain after further fine-tuning.

Because ProofWala unifies assistant-level execution, proof-state extraction, and annotated search-tree logging across Lean and Rocq, it enables not only aggregate evaluation (pass-at-k), but also fine-grained analysis of proof search dynamics. All experiments use the ProofWala-{Multilingual, Rocq, Lean} models inside the search module described in Section 3.2. We run proof search on the test splits defined in Table 1b for CompCert, MathComp, GeoCoq, CategoryTheory, and Lean.

5.1 Experimental Setup

Search procedure. We use ProofWala proof-step models for single-step tactic prediction and conduct end-to-end proof search via beam search. Candidate tactics are ranked using the negative log-likelihood assigned by the proof-step model, and this score is used as the primary heuristic for node expansion. We employ a timeout of `600` seconds for most experiments; for GeoCoq, we use a higher timeout of `1200` seconds due to longer proof scripts and higher tactic execution latency.

Parallel execution. All proof search experiments use the environment-pool abstraction of `itp-interface`. Each frontier state is mapped to one or more cloned assistant environments, allowing candidate tactics to be executed in parallel without reinitializing the proof context. Because environment instances retain assistant-level proof states, this design preserves semantic correctness while enabling scalable exploration. The abstraction is shared across Lean and Rocq backends (see Section 3.1 and Section 3.2).

Ablations and statistical testing. To study transfer, we compare monolingual models (ProofWala-Rocq, ProofWala-Lean) against the multilingual model (ProofWala-Multilingual) on each data-mix. We additionally report paired bootstrap significance testing for the gap between multilingual and monolingual variants where applicable.

5.2 Aggregate Results

Table 3 summarizes *pass-at-k* results for $1 \leq k \leq 5$ across all data-mixes. Overall, ProofWala-Multilingual matches or improves over monolingual baselines across both assistants, suggesting that multilingual training provides useful transfer signals for proof-step prediction and downstream proof search. We observe statistically significant improvements on the largest benchmark (Lean/Mathlib), while improvements in other datasets are smaller and not statistically significant, and should be interpreted as indicative trends.

To evaluate adaptation to a new domain, we further fine-tune ProofWala-Multilingual and ProofWala-Rocq on CategoryTheory. The multilingual model achieves substantially higher pass-at-k after fine-tuning, with statistically significant improvements over the Rocq-only baseline, indicating that multilingual pretraining can improve downstream adaptation even when the target domain is represented in only one assistant.

This behavior is consistent with prior work on multilingual mathematical reasoning. In particular, MMA [10] demonstrates that models trained on multiple formal languages achieve improved downstream performance compared to monolingual training. Our results extend this observation to the setting of proof-step prediction and neural-guided proof search.

5.3 Case Study: Cross-Lingual Transfer in Category Theory

To better understand the transfer gains in CategoryTheory, we examined Rocq theorems that the multilingual model successfully proved while the Rocq-only model failed. In many such cases, we found structurally analogous Lean theorems in the multilingual training data, particularly for categorical constructions such as adjunctions, monicity, currying, and evaluation morphisms. Because the framework extracts theorem metadata and proof traces across assistants in a unified format, we can systematically identify structurally analogous theorems between Rocq and Lean and analyze transfer behavior at the declaration level. Table 4 shows representative examples.

[11] Statistical significance is assessed using a paired bootstrap test; we report significance when $p_{\text{value}} < 0.05$.

Table 3 Pass-at-k results across data-mixes using ProofWala proof search guided by different proof-step models. ProofWala-Multilingual is trained jointly on Lean and Rocq data; monolingual baselines are trained only on their respective assistants. CategoryTheory results report further fine-tuning on that domain.

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)[11]
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-Rocq	**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-Rocq	**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-Rocq	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-Rocq-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 4 Representative CategoryTheory examples where the multilingual model succeeds but the Rocq-only model fails, alongside structurally analogous Lean theorems present in the multilingual training data.

Rocq Theorem	Lean Equivalent	Multilingual	Rocq-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)` $\mathsf{uncurry}(\nVdash_{A \Rightarrow X}) = (\mathsf{exp.ev}(A)).\mathsf{app}(X)$	✓	✗

This result is especially interesting in light of prior findings in multilingual NLP. For instance, XLM-R [5] shows that multilingual training can lead to negative transfer, particularly for low-resource languages, where out-of-language tokens may dilute the model's effectiveness. By contrast, we observe that even when training on mixed-language data (with $M > N$ tokens), our multilingual models consistently outperform monolingual counterparts trained on the smaller in-language subset (N tokens). This suggests that formal languages exhibit more meaningful structural alignment across ITPs, enabling productive cross-lingual transfer rather than suffering from the interference effects observed in natural language settings.

5.4 Proof Tree Analysis

Beyond pass-at-k, ProofWala exposes the internal search process via annotated proof trees (Figure 3). Each proof tree contains only compilable transitions, ensuring that every node corresponds to a valid proof state and every edge corresponds to a tactic that executes without error. Because proof states are obtained directly from the assistant after tactic execution, these trees reflect true semantic state transitions rather than simulated or approximate transitions.

Tree size and branching. Table 5 compares average proof tree statistics across data-mixes. Because the framework logs every successfully executed tactic and resulting proof state, we can quantify structural properties of search, including node count, edge count, and average branching factor. In most settings, trees produced by Multilingual are larger and include higher branching factors, suggesting that multilingual models propose a larger number of valid tactics per state.

Table 5 Average proof tree statistics across data-mixes. Proof trees contain only compilable edges and valid states.

Data-Mix	ProofWala Model	Avg. Proof Tree Stats		
		# Nodes	# Edges	Avg. Degree
Lean	Lean	3.989	3.536	1.536
	Multilingual	4.729	4.689	1.983
MathComp	Rocq	2.534	1.739	1.167
	Multilingual	2.576	1.822	1.207
GeoCoq	Rocq	15.358	14.457	1.180
	Multilingual	17.144	15.75	1.353
CompCert	Rocq	8.048	7.480	1.404
	Multilingual	8.318	8.200	1.584
CategoryTheory	Rocq-Cat-Theory	5.674	5.804	2.301
	Multilingual-Cat-Theory	7.056	7.130	2.193

Table 6 Average proof length (number of tactics) and average proof search time (seconds) across data-mixes. Proof lengths are comparable across models, while Multilingual often searches longer, consistent with broader exploration and larger proof trees.

Data Mix	ProofWala Model	Avg. Proof Length	Avg. Time (s)
Lean	Multilingual	2.0363	30.5296
	Lean	2.0679	20.7370
CompCert	Multilingual	4.7913	59.9424
	Rocq	5.0270	70.3282
MathComp	Multilingual	2.4940	8.9993
	Rocq	2.4759	7.5634
GeoCoq	Multilingual	9.2486	107.1999
	Rocq	10.6954	74.3914
CategoryTheory	Multilingual	3.4909	39.2182
	Rocq	3.0426	27.4105

Search time and proof length. Table 6 summarizes both proof length and search time across data-mixes. Proof lengths are broadly comparable across models, suggesting that performance differences are not driven by systematically longer proofs. However, Multilingual typically searches longer, which correlates with larger proof trees and broader exploration.

The ability to correlate tree size, branching factor, timeout usage, and final proof length is made possible by the framework's explicit tree representation and execution logging, which records timestamps, state transitions, and tactic outcomes for every search run.

5.5 Additional Benchmarking and Scalability

MiniF2F. We additionally evaluate ProofWala on the `MiniF2F` benchmark [29, 28]. We run the same search pipeline as in Section 5.1, using ProofWala-Multilingual and ProofWala-Lean for Lean proof search. On this benchmark, ProofWala-

MULTILINGUAL achieves `pass@5` of 26.23%, slightly exceeding PROOFWALA-LEAN (25.41%), indicating that multilingual training provides consistent gains even when evaluated on an out-of-distribution benchmark.

We emphasize that our goal is not to achieve state-of-the-art performance on MiniF2F. Instead, we use this benchmark as an *out-of-distribution (OOD)* evaluation to study generalization and transfer under a controlled setup. As a result, our absolute performance is lower than recent systems that rely on significantly larger models and additional infrastructure, but the relative improvement of the multilingual model over the monolingual baseline provides evidence of transfer under OOD conditions.

Scalability via parallel environment pools. To quantify the impact of parallel proof search, we vary the number of CPU workers used by our Ray-backed environment pool (see Section 3.2). Increasing parallelism from 8 to 20 CPUs improves MiniF2F `pass@5` from 22.54% to 26.23%, while reducing average proving time from 83.32s to 74.56s. These gains arise from evaluating more candidate tactics per frontier state and pruning unpromising branches earlier.

Such scaling is enabled by the environment-pool abstraction, which allows assistant instances to be cloned, filtered by proof state, and reused during search rather than restarted from scratch. This architectural design is central to PROOFWALA: it enables controlled, scalable experimentation on how neural guidance interacts with parallel search, independent of model scale.

5.6 Declaration-Level Analysis and Visualization

Beyond proof search, our framework exposes declaration-level metadata extracted from LEAN projects, including full dependency graphs, namespace information, file paths, and declaration types. To facilitate interactive analysis, we built a Streamlit-based *Lean Declaration Database Explorer* on top of the extracted SQLite database produced by `itp-interface`.

Custom SQL-based analysis. The explorer supports direct SQL queries over the declaration graph. For example, we can identify declarations with the highest number of dependencies or dependents, enabling structural analysis of core library components. This level of introspection is possible because our Lean integration extracts repository-wide declaration information – not only tactic-level proof traces.

Dependency graph visualization. The explorer also supports interactive visualization of dependency and dependent graphs up to configurable depth. Figure 6 shows examples of both custom-query analysis and graph-based dependency exploration.

Overall, these experiments illustrate that PROOFWALA is not merely a proof search system, but a unified research framework for neural-guided theorem proving. Beyond reporting pass-at-k metrics, our framework exposes assistant-level execution traces, maintains explicit environment state across parallel search branches, and logs fully annotated proof trees containing only compilable transitions.

By combining multi-assistant interaction (LEAN and ROCQ), environment pooling for scalable parallel search, and declaration-level metadata extraction, PROOFWALA enables detailed quantitative and qualitative analyses of search behavior. This includes measuring tree size, branching factors, proof multiplicity, execution latency, and structural transfer across assistants – analyses that are difficult to obtain from aggregate success rates alone. Such instrumentation transforms proof search from a black-box evaluation into a reproducible and analyzable experimental pipeline for studying neural-guided reasoning in interactive theorem provers.

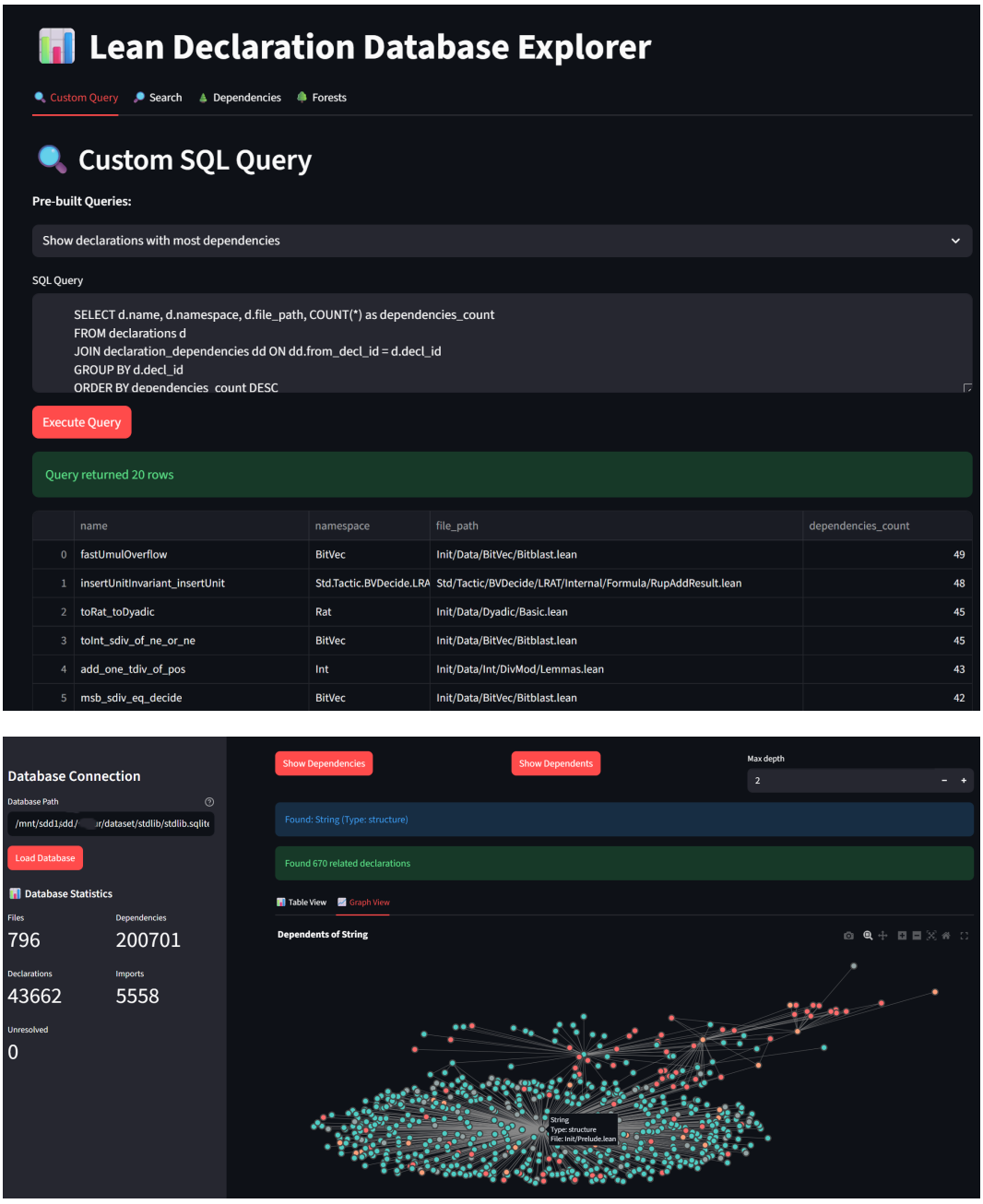

Figure 6 Streamlit-based Lean Declaration Database Explorer built on top of ProofWala's declaration extraction pipeline. **Top:** Custom SQL queries over the declaration database, for example identifying declarations with the largest number of dependencies. **Bottom:** Interactive dependency graph visualization for the `String` type, showing dependent declarations up to a configurable depth. Each node corresponds to a declaration, and additional information (name, type, file location) can be inspected interactively by hovering over nodes in the graph.

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 [16] for Lean, CompCert [13], and Mathcomp [15] for Rocq. LeanDojo [28] provided open-source tooling for interaction with Lean 3 and extracted a proof step dataset from Mathlib.[12] NTP Toolkit [30] supports extracting training data from arbitrary Lean repositories. CoqGym [26] is a framework for interaction and data collection with Rocq up to versions 8.12.0 (because of dependency on SerAPI library[13]). Proverbot [20] introduced Coq-Serapy, an interaction tool in Rocqfrom which our Rocqsupport is derived. CoqPyt [4] is a framework for interaction and data generation from Rocqwith emphasis on supporting LM-based methods. COPRA [23] introduces a framework for interaction with Lean 3 and Rocq, but without tooling for data extraction or support for heavy parallelism during proof search. Aniva et al. [1] introduced Pantograph, a Lean 4 interaction and proof tracing framework that supports tactic execution and proof-level data extraction. Pantograph enables collecting proof traces from Lean repositories and provides tooling for replaying and instrumenting proofs within Lean 4.

However, Pantograph is restricted to Lean and focuses primarily on tactic-level tracing. It does not provide cross-assistant abstraction, unified state representations, or repository-wide dependency analysis at the declaration level. In particular, it does not support extracting global dependency graphs, cross-file structural relationships, or multi-hop declaration dependencies in a form suitable for large-scale structural analysis or assistant-agnostic training.

In contrast, ProofWala is designed as a unified, cross-assistant framework supporting both Lean and Rocq. Beyond proof-step extraction, it enables declaration-level dependency graph construction, SQL-based repository exploration, standardized multilingual dataset generation, neural model training pipelines, and scalable parallel proof search via environment pooling. This architectural unification allows controlled cross-lingual transfer experiments and fine-grained structural analyses that extend beyond tactic replay or single-assistant tracing.

A number of proof search methodologies have been proposed in the recent literature. GPT-f [19] employed a best-first search approach with a trained transformer-based architecture for proof synthesis. LeanDojo [28] 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 [12, 7] introduces an online variant of Monte Carlo Tree Search for the theorem-proving task. PACT [8] introduces auxiliary training objectives derived from proof state data to learn a better prediction model for search. COPRA [23] 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 [2] learns online hierarchical representations of definitions and theorems, and is used for proof search in Tactician [3]. Several tools have been developed to help with live formalization efforts; these include LLMStep and LeanCopilot for Lean [25, 21], and CoqPilot for Rocq [11].

Previous work has explored providing effective support for measuring models across various interactive theorem provers. `miniF2F` [29] is a multi-language benchmark of high-school competition math problems formalized in Lean 3, HOL Light, Isabelle, and Metamath, though not in Rocq. PutnamBench [24] is a collegiate-level benchmark for competition

[12] LeanDojo now supports Lean 4, which is not backwards compatible to Lean 3.
[13] `https://github.com/rocq-archive/coq-serapi`

math in Lean 4, Rocq, and Isabelle. We do not include evaluations on PutnamBench as our work is not targeted towards olympiad-style theorem-proving. MMA [10] 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 ProofWala, a unified research framework for neural-guided theorem proving across interactive proof assistants such as Lean and Rocq. The framework standardizes assistant interaction, proof-state extraction, dataset construction, model training, and scalable parallel proof search under a common abstraction.

Using this infrastructure, we constructed a multilingual proof-step dataset spanning mathematics and software verification repositories and trained the first multi-assistant proof-step prediction models across Lean and Rocq. Our experiments demonstrate that multilingual training provides evidence of transfer across assistants and domains. We observe the strongest improvements on the largest benchmark and in the domain adaptation setting. At the same time, other datasets show consistent, albeit smaller, gains over monolingual baselines in both proof-step prediction and end-to-end proof search. This suggests that the benefits of multilingual training are most pronounced in high-data regimes and when adapting to new domains.

Beyond the empirical results, the primary contribution of this work is architectural. By unifying execution, data collection, dependency analysis, environment pooling, and search under a single framework, ProofWala enables systematic, reproducible, and fine-grained analysis of neural proof search across multiple ITPs. We view this unification as an essential step toward scalable and comparable research in automated theorem proving.

In future work, we aim to extend the framework to additional assistants and richer dependency representations, explore adaptive and learning-based search strategies within the unified abstraction, and investigate reinforcement learning approaches that leverage the logged proof-tree structures produced during search. We believe that such framework-centered development will be crucial for advancing neural-assisted formal reasoning.

References

1 Leni Aniva, Chuyue Sun, Brando Miranda, Clark Barrett, and Sanmi Koyejo. Pantograph: A machine-to-machine interaction interface for advanced theorem proving, high level reasoning, and data extraction in lean 4. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 104–123. Springer, 2025. doi:10.1007/978-3-031-90643-5_6.

2 Lasse Blaauwbroek, Miroslav Olšák, Jason Rute, Fidel Ivan Schaposnik Massolo, Jelle Piepenbrock, and Vasily Pestun. Graph2tac: Online representation learning of formal math concepts, 2024. arXiv:2401.02949.

3 Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. *The Tactician: A Seamless, Interactive Tactic Learner and Prover for Coq*, pages 271–277. Springer International Publishing, 2020. doi:10.1007/978-3-030-53518-6_17.

4 Pedro Carrott, Nuno Saavedra, Kyle Thompson, Sorin Lerner, João F. Ferreira, and Emily First. 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*, pages 637–641. ACM, July 2024. doi:10.1145/3663529.3663814.

5 Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019. arXiv:1911.02116.

6 Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 378–388. Springer, 2015.

7 Fabian Gloeckle, Jannis Limperg, Gabriel Synnaeve, and Amaury Hayat. 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=kk3mSjVCU0.

8 Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021. arXiv:2102.06203.

9 Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. The coq proof assistant a tutorial. *Rapport Technique*, 178, 1997.

10 Albert Q. Jiang, Wenda Li, and Mateja Jamnik. Multilingual mathematical autoformalization, 2023. doi:10.48550/arXiv.2311.03755.

11 Andrei Kozyrev, Gleb Solovev, Nikita Khramov, and Anton Podkopaev. Coqpilot, a plugin for llm-based generation of proofs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, pages 2382–2385. ACM, October 2024. doi:10.1145/3691620.3695357.

12 Guillaume Lample, Timothee Lacroix, Marie-Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. Hypertree proof search for neural theorem proving. *Advances in Neural Information Processing Systems*, 35:26337–26349, 2022.

13 Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009. doi:10.1145/1538788.1538814.

14 Zhaoyu Li, Jialiang Sun, Logan Murphy, Qidong Su, Zenan Li, Xian Zhang, Kaiyu Yang, and Xujie Si. A survey on deep learning for theorem proving, 2024. doi:10.48550/arXiv.2404.09939.

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

16 The mathlib Community. 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.

17 Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018.

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

19 Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020. arXiv:2009.03393.

20 Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. Generating correctness proofs with neural networks. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 1–10, 2020. doi:10.1145/3394450.3397466.

21 Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Towards large language models as copilots for theorem proving in lean, 2024. doi:10.48550/arXiv.2404.12534.

22 Streamlit Inc. Streamlit: A faster way to build and share data apps, 2026. Accessed: 2026-05-04. URL: https://streamlit.io/.

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

24 George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition, 2024. doi:10.48550/arXiv.2407.11214.

25 Sean Welleck and Rahul Saha. Llmstep: Llm proofstep suggestions in lean, 2023. doi:10.48550/arXiv.2310.18457.

26 Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning*, pages 6984–6994. PMLR, 2019. URL: http://proceedings.mlr.press/v97/yang19a.html.

27 Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in ai, 2024. doi:10.48550/arXiv.2412.16075.

28 Kaiyu Yang, Aidan M Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *arXiv preprint arXiv:2306.15626*, 2023. doi:10.48550/arXiv.2306.15626.

29 Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021. arXiv:2109.00110.

30 Thomas Zhu, Josh Clune, and Sean Welleck. Neural theorem proving toolkit, 2023. URL: https://github.com/cmu-l3/ntp-toolkit.