PROOFGYM: Unifying LLM-Based Theorem Proving Across Formal Systems

Xinrui Li[1], **Wenjie Ma**[1], **Hangrui Bi**[2], **Zhaoyu Li**[2], **Xujie Si**[2], **Kaiyu Yang**[3]
[1]UC Berkeley, [2]University of Toronto, [3]Meta FAIR
{henry_lxr89,windsey}@berkeley.edu, zhaoyu@cs.toronto.edu

Abstract

Large language models (LLMs) have accelerated progress in automated theorem proving, but most systems remain confined to a single proof assistant, hindering cross-system reuse of reasoning patterns and complicating scalable evaluation. We present PROOFGYM, a lightweight, high-throughput backend that unifies interaction with heterogeneous proof assistants (Coq, Isabelle, Lean) behind a common Python API. PROOFGYM supports both whole-proof and interactive step-wise modes, offers a language-agnostic state/result schema, enables non-blocking batched execution with bounded concurrency, and emits structured logs suitable for dataset curation and evaluator development. Preliminary experiments show substantial end-to-end throughput improvements for verification and proof search while preserving per-request latency. This paper focuses on system design, abstractions, and cross-system pipelines; full-scale training for a multi-language theorem proving model and broader ablations are left as ongoing work.

1 Introduction

LLM-based theorem proving has advanced rapidly, yet contemporary systems are largely siloed within individual formal languages. Across the three major proof assistants—Lean, Coq [8], and Isabelle/HOL [10]—machine learning (ML)-facing infrastructure has matured in complementary ways. In Lean, efforts such as LeanDojo [12] and the Kimina Lean server [6] provide large training corpora and high-throughput server interfaces. In Coq, ML work has coalesced around CoqGym [11]—a large learning environment and dataset of human proofs—and machine interfaces such as SerAPI and Coq-LSP [2, 7] that expose Coq's internals for data/interaction. For Isabelle, recent ML-facing infrastructure centers on PISA (Portal to Isabelle) [3], which provides a gRPC-based control plane for Isabelle, enabling programmatic orchestration of sessions, corpus extraction, and large-scale agent evaluation. These strands underscore our claim: despite vibrant progress in each community, tooling, data formats, and evaluators remain language-specific. This fragmentation (i) increases engineering overhead, (ii) impedes straightforward comparisons across languages, and (iii) limits the construction of standardized datasets and evaluators that could benefit from shared abstractions. A unified, high-throughput substrate for multi-language interaction is therefore a prerequisite for scalable research on cross-system reasoning.

We introduce PROOFGYM, a lightweight asynchronous backend that exposes a single Python API across multiple PAs. The API supports two complementary interaction modes: *whole-proof* (batch verification of files/chunks) and *interactive* (stepwise command/tactic execution). Under the hood, language-specific workers are orchestrated by an async server with bounded concurrency. A language-agnostic data model (`ProofState`, `VerificationResult`, `Status`) normalizes heterogeneous REPL outputs while preserving assistant-specific metadata for downstream use. The server emits structured, per-step logs to facilitate evaluation, curation, and reward-signal construction.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

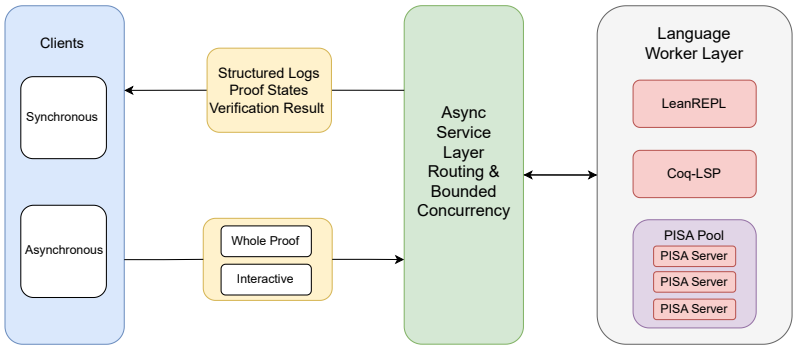

Figure 1: PROOFGYM architecture: clients → async service → language workers → results.

Standardizing execution and state representation decouples proving algorithms from proof assistant idiosyncrasies, making experiment code portable and enabling rigorous comparisons under shared budgets. More importantly, multi-assistant interaction becomes a *data product*: aligned trajectories, outcomes, and diagnostics can be aggregated at scale, supporting reliable evaluator development and cross-system analysis.

Preliminary evidence and scope. In initial experiments, PROOFGYM provides sizable throughput gains for both whole-proof verification and interactive proof search without degrading per-request latencies, primarily due to non-blocking request handling and lean worker management. In this paper, we emphasize the system and abstractions rather than final model performance; broader training, cross-assistant alignment, and exhaustive ablations are ongoing work.

2 Method

Our framework, PROOFGYM, is a lightweight system designed for robust, large-scale interaction with multiple theorem provers. As shown in Figure 1, its architecture follows a "client—async service—language worker pool" pattern. The core of our method is a unified API that abstracts away prover-specific complexities, enabling uniform interaction with Lean 4, Coq, and Isabelle.

2.1 A Unified API for Formal Verification

The central abstraction in PROOFGYM is a minimal, portable API built upon a language-agnostic schema and two primary modes of operation.

Language-Agnostic Schema. All interactions use a shared data model. The core objects are the `ProofState`, which represents the current goals, context, and cursor location, and the `VerificationResult`, which encapsulates the outcome (*e.g.*, success or failure), diagnostics, and a trajectory of states. Prover-specific details, like Lean metavariables or Coq hypothesis formatting, are preserved in optional metadata fields, ensuring high-fidelity data without sacrificing portability.

Operation Modes. The API supports two modes:

- **Whole-Proof**: Verifies a complete file or code block in a single call, returning a final `VerificationResult` with the full trajectory. This is suitable for batch verification tasks.
- **Interactive**: Executes proof steps incrementally. Each call takes a `ProofState` and a list of commands, returning a list of new `ProofState` trajectories. This makes the server stateless, as the `ProofState` acts as the sole resume token, simplifying state management for clients.

This design is exposed through simple API endpoints like `verify`(file) and `apply_steps`(state,steps), available via both synchronous and asynchronous Python clients.

Table 1: Per-language capabilities exposed via the unified API.

Capability	Lean	Coq	Isabelle
Whole-proof execution	✓	✓	✓
Interactive stepping	✓	✓	✓
State save/fork/resume	✓	–	–
Project-aware caching	–	–	✓

2.2 System Architecture and Implementation

Async Service Layer. The heart of the system is an asynchronous server built with FastAPI. This layer is intentionally thin, with its primary responsibilities being request routing and concurrency management. It uses a bounded semaphore to manage the lifecycle of language worker processes, creating and destroying them on demand without leaving workers idle. All interactions through the service generate structured logs containing trajectories and diagnostics processed from outputs of language workers, which are invaluable for debugging, evaluation, and curating datasets.

Language Worker Pools. For each prover, PROOFGYM uses a dedicated worker implementation to interface with the underlying tool.

- **Lean**: We wrap `LeanREPL` [4], which supports both whole-proof execution and interactive stepping. Crucially, it allows the proof environment to be pickled, enabling a state to be saved and then efficiently forked across multiple REPLs without replaying the entire proof history.
- **Coq**: Our worker wraps Coq-LSP, normalizing its JSON-RPC output to conform to the PROOFGYM schema while retaining Coq-specific diagnostics in metadata fields.
- **Isabelle**: We use the Portal-to-Isabelle (PISA) server [3]. Due to PISA's significant initialization cost, we maintain a persistent pool of PISA servers. A dynamic, LRU-style policy reuses workers for requests that share the same project context and imports, amortizing the startup overhead.

As summarized in Table 1, this architecture allows us to expose both common and prover-specific capabilities through a single, consistent interface. While Lean and Coq sessions are ephemeral, the server remains stateless by design, as no request permanently reserves a worker.

3 Experiments

3.1 Experimental Setup

We evaluate PROOFGYM in Lean and Coq on two tasks: whole-proof verification and proof search against ProofWala's ITP-Interface [9], which is the only open multi-proof-assistant framework that supports both tasks. For verification, we use 100 problems from Lean's `GoedelLeanWorkbook` [5] and 100 lemmas of varying proof length from Coq's `GeoCoq` [1]. For proof search, we use pre-generated beams (width 32) on Lean's MiniF2F test set (244 problems) and the GeoCoq test set (478 lemmas). All experiments were run on an 80-CPU machine with timeouts of 120s (verification) and 600s (search). We cap simultaneous processes in PROOFGYM (`MAX_CONCURRENT_REQUESTS`) and ITP-Interface (`max_parallel_env`) to ensure a fair throughput comparison.

3.2 Results

Whole-Proof Verification. As shown in Table 2, PROOFGYM demonstrates substantial throughput improvements for batch verification. On the combined Lean and Coq datasets, PROOFGYM completes the task in just 68 seconds, a 7.4-fold speedup over the Ray-parallelized ITP-Interface baseline (505s). The performance gains are even more pronounced on individual datasets: we observe a 4.6× speedup on Lean's `GoedelLeanWorkbook` and a remarkable 31.2× speedup on Coq's `GeoCoq`. These gains are attributable to PROOFGYM's core design. The asynchronous server architecture with bounded concurrency prevents long-running proofs from blocking the entire worker pool, improving resource utilization. Furthermore, our lightweight, ephemeral REPL management avoids the complex orchestration overhead inherent in frameworks that rely on persistent, stateful worker pools.

Table 2: Whole-proof verification. PROOFGYM substantially reduces total wall time while preserving per-proof latencies.

Dataset & Scope	Framework	Configuration	Total (s)	Avg (s)	Max (s)
GoedelLeanWorkbook (100 Lean problems)	ITP-Interface	No Ray; step-wise (with proof-state data)	1350.41	13.50	—
	ITP-Interface	No Ray; whole-proof	1095.68	10.96	—
	ITP-Interface	Ray; `max_parallel_env`=32; whole-proof	286.997	—	—
	PROOFGYM	`MAX_CONCURRENT`=**32**	**63**	**12.12**	**53.88**
GeoCoq OriginalProofs (first 100 lemmas)	ITP-Interface	No Ray; whole-proof	439.594	4.40	—
	ITP-Interface	Ray; `max_parallel_env`=32; whole-proof	249.919	—	—
	PROOFGYM	`MAX_CONCURRENT`=**32**	**8**	**1.026**	**5.315**
Lean+Coq (combined)	ITP-Interface	Ray; `max_parallel_env`=32; whole-proof	505.360	—	—
	PROOFGYM	`MAX_CONCURRENT`=**32**	**68**	—	—

Table 3: Proof-search wall time (beam width 32). PROOFGYM benefits strongly from asynchronous clients and higher concurrency.

Dataset & Framework	Max parallel / concurrent	Client mode	Total (s)
Lean MiniF2F (244 problems)			
Proofwala	Max_Parallel = 32	—	8482.02
Proofwala	Max_Parallel = 64	—	7027.51
PROOFGYM	**MAX_CONCURRENT = 32**	**Synchronous**	**3801.33**
PROOFGYM	**MAX_CONCURRENT = 32**	**Asynchronous**	**1202.29**
PROOFGYM	**MAX_CONCURRENT = 64**	**Asynchronous**	**962.04**
Coq GeoCoq (478 problems)			
Proofwala	Max_Parallel = 32	—	130,530.11
Proofwala	Max_Parallel = 64	—	65,507.01
PROOFGYM	**MAX_CONCURRENT = 32**	**Synchronous**	**64,215.84**
PROOFGYM	**MAX_CONCURRENT = 32**	**Asynchronous**	**35,916.65**
PROOFGYM	**MAX_CONCURRENT = 64**	**Asynchronous**	**32,855.79**

Proof search. In the more demanding proof search task (Table 3), the benefits of PROOFGYM's design are more pronounced. We compare ProofWala against PROOFGYM using both synchronous and asynchronous clients to isolate the impact of client-side scheduling. With an asynchronous client that can fully leverage the server's capacity, PROOFGYM achieves end-to-end speedups of 7.3× on Lean (962s vs. 7028s) and 2.0× on Coq (32,856s vs. 65,507s) compared to the fastest ProofWala configuration. Notably, client-side asynchrony is a key performance driver. At the same concurrency level of 32, simply switching the PROOFGYM client from synchronous to asynchronous improves throughput by 3.2× on Lean. This highlights a critical bottleneck in typical proof search pipelines: without an asynchronous client, server-side resources can be left idle while the client sequentially processes requests. Even with a synchronous client, PROOFGYM's leaner architecture provides a $\approx 2\times$ speedup, but unlocking the full potential requires an end-to-end asynchronous design.

Discussion. Our experiments highlight three key findings. First, a fully asynchronous server architecture is crucial for maximizing throughput in batch-processing workloads. By decoupling request admission from worker execution, the system can better tolerate high-variance task runtimes. Second, a lightweight server and REPL management design can significantly reduce orchestration overhead, yielding substantial performance gains even at moderate levels of parallelism. Third, for iterative tasks like proof search, client-side asynchrony is as important as server-side concurrency. A synchronous client can become the primary bottleneck, preventing the system from saturating available computational resources. Finally, the ability to fork proof state (checkpointing and branching from intermediate proof contexts) significantly eliminates the overhead of replaying history during long proof search tasks, enabling fast backtracking and broader concurrent exploration. PROOFGYM's design addresses all four aspects, delivering significant efficiency improvements.

4 Conclusion and Ongoing Work

We presented PROOFGYM, a unified, high-performance framework for Lean, Coq, and Isabelle that achieves up to a $31\times$ speedup over prior work on verification and proof search tasks. Our results show that its fully asynchronous client-server architecture is critical for eliminating bottlenecks in large-scale automated reasoning, providing a robust foundation for future research.

Building on this platform, our ongoing work aims to develop a multilingual proof synthesis pipeline. We have already generated a corpus of **150k** parallel informal-formal statements across the three languages (50k each) and are training a 7B model for statement formalization and cross-system translation. The goal is to produce aligned `⟨statement, proof⟩` triplets across all three systems, enabling large-scale supervised fine-tuning of a unified, multilingual theorem prover.

References

[1] Gabriel Braun, Pierre Boutry, Charly Gries, Julien Narboux, Pascal Schreck, and GeoCoq contributors. Geocoq: A formalization of foundations of geometry in Coq. `https://geocoq.github.io/GeoCoq/`, 2024. Software, version 2.5.0 (`coq-geocoq`).

[2] 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*, pages 637–641, 2024.

[3] Albert Q. Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. Lisa: Language models of isabelle proofs. *6th Conference on Artificial Intelligence and Theorem Proving*, 2021.

[4] Lean FRO. A read-eval-print-loop for Lean 4. `https://github.com/leanprover-community/repl`, 2023. Software project.

[5] Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source automated theorem proving, 2025. URL `https://arxiv.org/abs/2502.07640`.

[6] Marco Dos Santos, Haiming Wang, Hugues de Saxcé, Ran Wang, Mantas Baksys, Mert Unsal, Junqi Liu, Zhengying Liu, and Jia Li. Kimina lean server: Technical report, 2025. URL `https://arxiv.org/abs/2504.21230`.

[7] Coq LSP Development Team. Coq lsp. URL `https://github.com/ejgallego/coq-lsp`. Software release.

[8] The Rocq Development Team. The rocq prover, April 2025. URL `https://doi.org/10.5281/zenodo.15149629`.

[9] Amitayush Thakur, George Tsoukalas, Greg Durrett, and Swarat Chaudhuri. proofwala: Multilingual proof data synthesis and theorem-proving, 2025. URL `https://arxiv.org/abs/2502.04671`.

[10] Makarius Wenzel and Isabelle contributors. *The Isabelle/Isar Reference Manual*, 2025. URL `https://isabelle.in.tum.de/doc/isar-ref.pdf`. Version <your-version>.

[11] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning (ICML)*, 2019.

[12] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. In *Neural Information Processing Systems (NeurIPS)*, 2023.