PROOFBRIDGE: AUTO-FORMALIZATION OF NATURAL LANGUAGE PROOFS IN LEAN VIA JOINT EMBEDDINGS

Anonymous authors
Paper under double-blind review

ABSTRACT

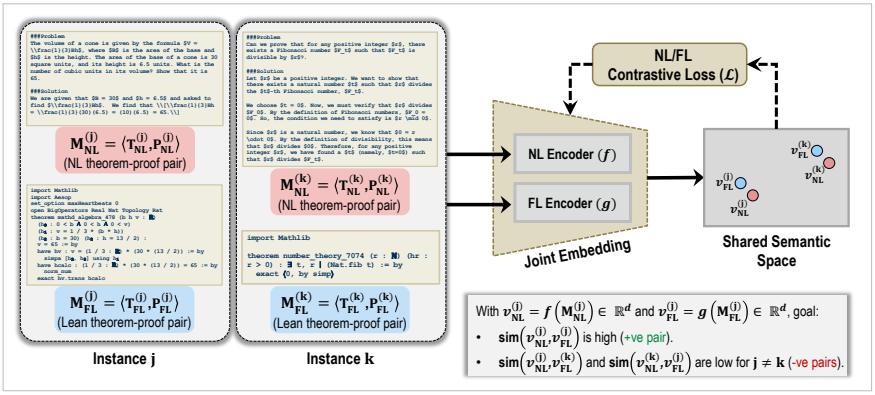
Translating human-written mathematical theorems and proofs from natural language (NL) into formal languages (FLs) like Lean 4 has long been a significant challenge for AI. Most state-of-the-art methods address this separately, first translating theorems and then generating proofs, creating a fundamental disconnect vis-a-vis true proof auto-formalization. This two-step process and its limitations were evident even in AlphaProof's silver-medal performance at the 2024 IMO, where problem statements needed manual translation before automated proof synthesis.

We present PROOFBRIDGE, a unified framework for automatically translating entire NL theorems and proofs into Lean 4. At its core is a joint embedding model that aligns NL and FL (NL-FL) theorem-proof pairs in a shared semantic space, enabling cross-modal retrieval of semantically relevant FL examples to guide translation. Our training ensures that NL-FL theorems (and their proofs) are mapped close together in this space if and only if the NL-FL pairs are semantically equivalent. PROOFBRIDGE integrates retrieval-augmented fine-tuning with iterative proof repair, leveraging Lean's type checker and semantic equivalence feedback to ensure both syntactic correctness and semantic fidelity. Experiments show substantial improvements in proof auto-formalization over strong baselines (including GPT-5, Gemini-2.5, Kimina-Prover, DeepSeek-Prover), with our retrieval-augmented approach yielding significant gains in semantic correctness (SC, via proving bi-directional equivalence) and type correctness (TC, via type-checking theorem+proof) across pass@k metrics on MINIF2F-TEST-PF, a dataset we curated. In particular, PROOFBRIDGE improves cross-modal retrieval quality by up to $3.28\times$ Recall@1 over all-MiniLM-L6-v2, and achieves +31.14% SC and +1.64% TC (pass@32) compared to the baseline Kimina-Prover-RL-1.7B.

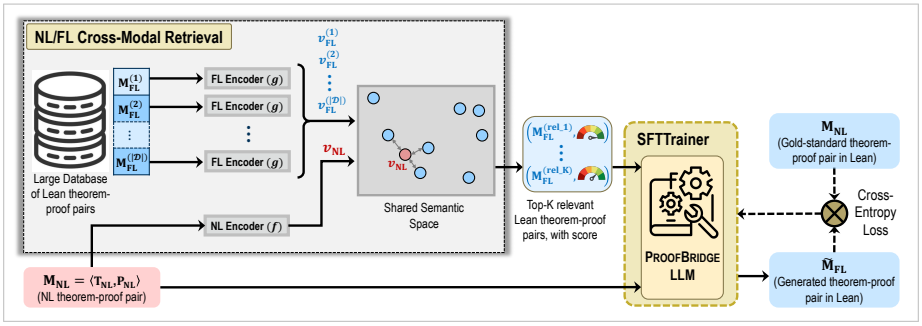
1 INTRODUCTION

In mathematics, ensuring the correctness of proofs is a crucial yet inherently difficult task. Traditionally, mathematicians rely on the peer-review process for *proof verification*, yet as proofs grow increasingly complex, even careful human scrutiny can overlook subtle errors. For instance, in 1989, Kapranov and Voevodsky published a proof connecting ∞-groupoids and homotopy types, which was later disproven by Carlos Simpson in 1998; more recently, while formalizing his 2023 paper (Tao, 2023) on the Maclaurin-type inequality, Terence Tao discovered a non-trivial bug. To mitigate challenges of verifying complex proofs, proof assistants and formal mathematical languages like Coq (Barras et al., 1999), Isabelle (Nipkow et al., 2002), HOL Light (Harrison, 2009), Metamath (Megill & Wheeler, 2019), Lean 4 (Moura & Ullrich, 2021), and Peano (Poesia & Goodman, 2023) have been developed, offering a way to create *computer-verifiable formal proofs*. Such formal language (FL) proofs, defined by strict syntax and symbolic logic, enable reliable automated verification guarantees that resolve the inherent ambiguity of natural language (NL) proofs. However, constructing FL proofs is time-intensive and demands both deep mathematical expertise and detailed knowledge of the language and its libraries, making the process challenging even for experienced mathematicians and limiting the wider adoption of such theorem provers and FL proofs.

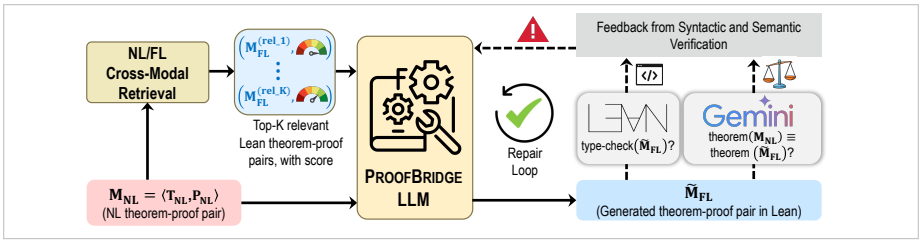
To simplify the task of writing proofs in FL, two key research directions have emerged: *auto-formalization* and *automated formal proof synthesis*. *Auto-formalization* targets NL-to-FL translation, but most prior works (Wang et al., 2025; Wu et al., 2025; Jiang et al., 2024; Gao et al., 2024)



(a) Joint embedding of NL and FL (Lean) theorems and proofs into shared semantic space



(b) Retrieval-augmented Supervised Fine-Tuning (SFT) of PROOFBRIDGE with NL/FL cross-modal retrieval



(c) Inference phase of retrieval-augmented proof auto-formalization with iterative repair

Figure 1: **Pipeline of PROOFBRIDGE for proof auto-formalization.** We first train a joint embedding model for NL and FL via contrastive learning, enabling cross-modal retrieval of semantically related FL theorem-proof pairs for a given NL input. An LLM is then fine-tuned on NL-to-Lean translations, conditioned on retrieved proofs and relevance scores. At inference, the system retrieves related Lean proofs and applies an iterative repair loop to the generated FL theorem-proof pair.

focus only on formalizing theorems (statements), not proofs. In contrast, *automated formal proof synthesis* (Ren et al., 2025; Wang et al., 2025) aims to generate FL proofs given an FL theorem. Proof auto-formalization is relatively less explored, with Draft-Sketch-Prove (Jiang et al., 2022) for Isabelle and FormL4 (Lu et al., 2024) for Lean serving as notable examples. In practice, formalizing an entire NL proof requires first performing *theorem auto-formalization* to translate the NL theorem into FL, followed by *automated formal proof synthesis* to generate the FL proof from the FL theorem. AlphaProof (Deepmind, 2024), which achieved silver-medal standard in the 2024 International Mathematical Olympiad, followed this two-step process: problems were first manually translated into formal mathematical language, then formal proofs were synthesized. This highlights the disconnect in formalizing NL proofs with current methods.

Contemporary LLMs face several challenges that limit their effectiveness for proof auto-formalization in Lean 4. **First**, large-scale datasets pairing NL theorems with Lean 4 proofs are scarce. Most existing resources (Goedel-Pset-v1 (Lin et al., 2025), Herald statements (Gao et al., 2024), Lean Workbook (Ying et al., 2024), MMA (Jiang et al., 2024)) cover only theorems, while

those with proofs (Herald proofs, Lean Workbook proofs (Lin et al., 2025), and FormL4 (Lu et al., 2024)) are much smaller and do not align with the popular miniF2F (Zheng et al., 2021) benchmark in the same Lean 4 version. Lean versions are not backward compatible, so cross-version evaluation often fails. **Second**, most fine-tuned LLMs for Lean 4 target either theorem auto-formalization or proof synthesis. Proof auto-formalization is harder, as it requires both translating the NL theorem and constructing the corresponding FL proof. **Third**, Lean 4 has an effectively *infinite action space* (Poesia & Goodman, 2023), with proofs using complex *tactics* that reuse prior theorems or introduce new variables. Prior work generates FL directly from NL, ignoring semantic relations like shared tactics and DAG dependencies, causing LLMs to often violate Lean 4's strict syntactic and semantic constraints and produce hallucinated or invalid proofs (Jha et al., 2025; Jana, 2024; Ugare et al., 2024). **Fourth**, automated evaluation is a major bottleneck. Lean's type-checker verifies the FL proof but cannot ensure semantic equivalence. Existing methods often type-check only the theorem (leaving proofs incomplete using placeholders like `sorry`) or rely on proxies such as BLEU, which are unreliable (Jiang et al., 2024; Lu et al., 2024; Wu et al., 2022; Ying et al., 2024).

Key Insight. In this paper, we address the task of *proof auto-formalization*, focusing on Lean 4 as the FL, via a combination of a joint embedding model, an LLM, and Lean for verification. It takes as input an NL theorem-proof pair and produces the corresponding FL theorem-proof pair in Lean 4. The key insight behind our approach is to treat proof auto-formalization as *learning from demonstrations*: the LLM is guided not only by the NL proof but also by FL proofs retrieved using an NL/FL joint embedding model that leverages contrastive learning and encodes linear DAG traversals of Lean proofs. Rather than relying on randomly chosen few-shot examples, these retrieved proofs capture far richer *reusable formalization patterns* (tactic choices, DAG structures), providing grounded signals that guide generation toward Lean-verifiable proofs, as illustrated in Figure 1.

Contributions:

The PROOFBRIDGE Auto-formalization Method and Tool: We present PROOFBRIDGE, an LLM-based, retrieval-augmented proof auto-formalization framework. At its core is an *NL/FL joint embedding model* that maps semantically equivalent NL and FL theorem-proof pairs to nearby points in a shared space, enabling effective cross-modal retrieval of related FL proofs. We then fine-tune the SoTA LLM Kimina-Prover-RL-1.7B (Wang et al., 2025) to perform NL-to-Lean 4 proof translation, conditioned on the retrieved FL proofs and their relevance scores. During inference, generation is refined with an iterative verifier-guided repair loop combining Lean type-checking with LLM-based bi-directional equivalence proving to ensure syntactic correctness and semantic fidelity. (Section 4)

NuminaMath-Lean-PF Dataset: We curate NUMINAMATH-LEAN-PF, a large-scale dataset of 38.9k NL↔Lean 4 theorem-proof pairs, specialized for *proof auto-formalization*. Each Lean theorem-proof pair is type-checked and paired with an NL counterpart. Additionally, we release MINIF2F-TEST-PF, a Lean v4.15.0 version of miniF2F-Test with 244 instances tailored for proof auto-formalization, enabling a consistent pipeline in the same Lean version. (Section 5.1)

Extensive Experimental Evaluation: Compared to the SoTA encoder all-MiniLM-L6-v2, PROOFBRIDGE's cross-modal NL→FL retrieval achieves $3.28\times$ higher Recall Rate@K at K=1 and $2.74\times$ MRR, with top-K retrieved embeddings 23% closer and non-retrieved 104% farther. We evaluate PROOFBRIDGE against 10 SoTA LLMs, including foundation models (Gemini-2.5, GPT-5-mini) and automated proof synthesis LLMs (DeepSeek-Prover, STP, Leanabell-Prover, Kimina-Prover), using verifier-grounded metrics: *type correctness* (TC) and *semantic correctness* (SC, a new metric based on Lean bidirectional equivalence proofs). Built on Kimina-Prover-RL-1.7B, PROOFBRIDGE achieves +31.14% SC and +1.64% TC (pass@32) on MINIF2F-TEST-PF. (Section 5)

2 RELATED WORK

Our work lies at the intersection of three key AI-for-Math research areas: automated formal proof synthesis, auto-formalization, and retrieval-augmented learning for mathematical reasoning. We focus on the most relevant approaches and highlight differences from our unified framework.

Auto-Formalization. Auto-formalization translates NL mathematics into FL, but most existing work focuses on theorem formalization rather than proofs. **Theorem-only approaches** include Herald-translator (Gao et al., 2024), which extracts FL theorems from Mathlib4 and trains on informal counterparts, and Kimina-Autoformalizer (Wang et al., 2025), which fine-tunes models with expert iteration. These excel at translating theorems but cannot handle proofs. **Proof auto-**

formalization has received limited attention. Draft-Sketch-Prove (Jiang et al., 2022) converts NL proofs into formal sketches in Isabelle with open conjectures, then fills gaps using predefined tactics and tools like Sledgehammer (Paulsson & Blanchette, 2012). FormL4 (Lu et al., 2024) trains on GPT-4 informalized Mathlib proofs with process-supervised step-level Lean compilation feedback.

Key Differences: Our approach is the first to jointly learn representations for NL and FL theorem-proof pairs, enabling cross-modal retrieval to guide formalization. Unlike prior work on isolated proof generation, we leverage semantic relationships of NL and FL proofs for contextual guidance.

Automated Formal Proof Synthesis. Automated formal proof synthesis (AFPS) takes FL theorems as input and generates FL proofs. Current approaches fall into two categories: *next-tactic prediction* and *whole-proof generation*. **Next-Tactic Prediction (NTP)** methods train models to predict single proof steps from current proof states. Representative systems include GPT-f (Polu & Sutskever, 2020) for Metamath, LIsa (Jiang et al., 2021) for Isabelle, and PACT (Han et al., 2022) for Lean. These use tree search over proof states, prioritizing tactics by cumulative probability. While NTP ensures stepwise correctness via interactive theorem-prover verification, it suffers from long-horizon dependencies and computational overhead from such repeated interactions. **Whole-Proof Generation (WPG)** methods generate complete FL proofs in single passes, offering computational efficiency but risking cascading errors. Recent advances include DeepSeek-Prover-v1 (Xin et al., 2024a), which combines SFT with expert iteration, and TheoremLlama (Wang et al., 2024), which improves in-context learning through curriculum-based training. DeepSeek-Prover-v2 (Ren et al., 2025) integrates NL reasoning with formal proof generation, while Kimina-Prover (Wang et al., 2025) applies reinforcement learning with compilation-based rewards (Jana et al., 2024).

Key Differences: Unlike AFPS approaches that assume FL theorems as input, our work addresses the more challenging task of generating theorem-proof pairs in FL from an NL input.

Retrieval-Augmented Learning for Mathematics. Recent work has explored retrieval-augmented approaches for mathematical reasoning, though not specifically for auto-formalization. TLAPS (Zhou, 2025) retrieves previously verified proofs to assist with proof generation, while COPRA (Thakur et al., 2023) uses retrieval to select relevant lemmas and guide proof search.

Key Differences: Our joint embedding approach enables cross-modal retrieval between natural language and formal theorems, which is essential for proof auto-formalization but not addressed by existing retrieval-augmented mathematical reasoning systems.

Positioning our contributions. PROOFBRIDGE makes several novel contributions relative to existing approaches: *(a) Unified Proof Auto-Formalization:* We address complete translation (theorem + proof) rather than treating theorem formalization and proof synthesis separately. *(b) Joint Semantic Embedding:* Our contrastive learning framework for aligning NL and FL proofs is novel, enabling effective cross-modal retrieval. *(c) Retrieval-Augmented Translation:* We are the first to apply retrieval-augmented fine-tuning and generation to auto-formalization, leveraging semantic relationships between FL proofs to guide translation. *(d) Rigorous Evaluation:* We introduce systematic metrics for proof auto-formalization, including type correctness via bi-directional equivalence rather than proxy measures. This combination of joint embedding, retrieval augmentation, and unified translation distinguishes our approach from prior work.

3 PRELIMINARIES: TACTIC-STYLE PROOFS IN LEAN

Lean (Moura & Ullrich, 2021) is a functional programming language and interactive theorem prover that is based on the propositions-as-types principle, where proving a proposition is equivalent to constructing a term of the corresponding type. Rather than building these terms manually, users write proofs in a tactic language, which provides high-level steps to guide term construction. Lean 4 (henceforth Lean) represents tactic-style proofs as directed acyclic graphs (DAGs) of *proof states* and *tactics*, automatically generating the corresponding proof term in the background. The kernel then verifies the term, ensuring correctness by enforcing the axiomatic foundations of Lean's logic, the Calculus of Inductive Constructions. This combination of a formal system and a small, trusted kernel provides strong confidence in the validity of proofs. In the DAG (Figure 2) of a Lean proof:

- Each **proof state** $S_i \equiv [G_1, \cdots, G_n]$ consists of a sequence of zero or more *open goals*. Initial state S_0 has one goal, the theorem $T_{FL} \equiv pr \vdash cn$ itself. Leaf-level states have no open goal.
- Each **open goal** $G_i \equiv pr_i \vdash cn_i$ of a proof state represents a proposition cn_i that needs to be proven, given a set of premises pr_i.

- Each **tactic** tac_i represents a proof step. It is a high-level command (rooted in metaprogramming) applied to an open goal G_i, producing a new proof state. If the resulting proof state has no open goal, it directly resolves the current goal. A parent goal is resolved once all subgoals are resolved.

Tactic-style proof synthesis in Lean follows a *sequential decision process*. Lean provides an interactive REPL (Leanprover, 2025) that applies tactics step by step to manipulate proof states. An FL proof is a sequence of tactics, and at each step, the REPL updates the proof state if the tactic is valid or returns an error identifying the faulty one. Each tactic advances the proof by breaking the current goal into simpler subgoals, similar to the '*suffices to show*' construct.

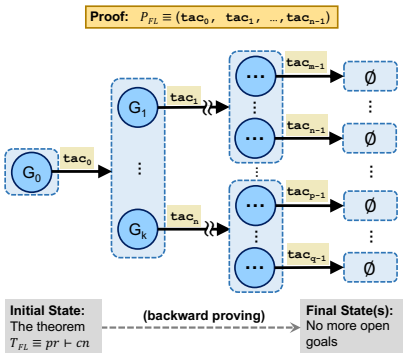


Figure 2: **Tactic-style Proof.** A Lean proof represented as a DAG of tactics.

Proof-Autoformalization as a Learning Problem. Given an NL theorem–proof pair $M_{\text{NL}} = \langle T_{\text{NL}}, P_{\text{NL}} \rangle$, the goal is to learn a function $f : M_{\text{NL}} \mapsto M_{\text{FL}}$ that produces a corresponding Lean theorem–proof pair $M_{\text{FL}} = \langle T_{\text{FL}}, P_{\text{FL}} \rangle$. Here, $T_{\text{FL}} \equiv pr \vdash cn$ denotes the formal theorem, and $P_{\text{FL}} \equiv (\texttt{tac}_0, \ldots, \texttt{tac}_{n-1})$ is the proof as a sequence of tactics. The generated pair must satisfy:

(a) *Type correctness:* $M_{\text{FL}} = \langle T_{\text{FL}}, P_{\text{FL}} \rangle$ passes Lean type-checking, ensuring that P_{FL} proves T_{FL} with no open goals in the DAG.
(b) *Semantic correctness:* FL theorem is semantically equivalent to the NL one, i.e., $T_{\text{FL}} \equiv T_{\text{NL}}$.

4 Our Approach and Tool Architecture

4.1 Joint Embedding of NL and Lean Proofs for Cross-Modal Retrieval

A core component of our framework is the *joint embedding model*, which learns to represent NL theorem–proof pairs and their FL (Lean) counterparts in a shared semantic space. The goal is to align these modalities so that cross-modal retrieval between NL and FL becomes possible. Formally, given an NL theorem-proof pair M_{NL} and a large database of FL theorem-proof pairs $\mathcal{D} = \left\{ M_{\text{FL}}^{(i)} = \langle T_{\text{FL}}^{(i)}, P_{\text{FL}}^{(i)} \rangle \right\}$, the model retrieves subset $\mathcal{R}(M_{\text{NL}}, \mathcal{D}) \subseteq \mathcal{D}$ of size $K \ll |\mathcal{D}|$, that serve as in-context demonstrations to guide downstream auto-formalization.

During training the joint embedding model, we start with NL-FL pairs $\left(M_{\text{NL}}^{(i)}, M_{\text{FL}}^{(i)}\right)$, which are encoded into vectors using two modality-specific encoders. Each encoder is initialized with a pre-trained model, and a subset of parameters is subsequently fine-tuned. Given $M_{\text{NL}}^{(i)}$, the NL encoder f produces an embedding $v_{\text{NL}}^{(i)} = f(M_{\text{NL}}^{(i)}, \theta_f \| \phi_f) \in \mathbb{R}^d$, and given $M_{\text{FL}}^{(i)}$, the FL encoder g produces $v_{\text{FL}}^{(i)} = g(M_{\text{FL}}^{(i)}, \theta_g \| \phi_g) \in \mathbb{R}^d$, where θ denotes frozen parameters, ϕ denotes trainable parameters, and d is the dimension of the shared semantic space. The details of each encoder are as follows:

- **NL encoder $f(M_{\text{NL}}^{(i)}, \theta_f \| \phi_f)$:** To encode $M_{\text{NL}}^{(i)}$, we use `all-MiniLM-L6-v2` (Reimers & Gurevych, 2020), a lightweight model (22.7M parameters) that effectively captures semantic similarity in NL. It encodes $M_{\text{NL}}^{(i)}$ into 384-dimensional embeddings, thereby projected into the joint embedding space of dimension $d = 512$ via a linear layer included in the trainable set ϕ_{f}.
- **FL encoder $g(M_{\text{FL}}^{(i)}, \theta_g \| \phi_g)$:** Given $M_{\text{FL}}^{(i)} = \langle T_{\text{FL}}^{(i)}, P_{\text{FL}}^{(i)} \rangle$, we first extract the linearized DAG traversal of tactics from $P_{\text{FL}}^{(i)}$ using Lean REPL (Leanprover, 2025). This traversal is represented as an ordered sequence of proof-state transformations induced by successive tactic applications: $S_0 \xrightarrow{\texttt{tac}_0} S_1 \xrightarrow{\texttt{tac}_1} \cdots \xrightarrow{\texttt{tac}_{H-1}} S_H$, where $S_0 \equiv T_{\text{FL}}^{(i)}$, each $S_h \equiv [G_1, \ldots, G_l]$ denotes a proof state consisting of zero or more open goals, and $\texttt{tac}_{h-1}$ is the tactic applied at step h. This sequence captures the entire proof as an ordered series of state transformations. To create embeddings for the full proof, we first encode each state S_h using LeanDojo's `ByT5` proof-state encoder (Yang et al., 2023) (218M parameters), producing embeddings of size 1,472 per state. We then obtain a single embedding for the entire proof via mean-pooling, which is subsequently projected into a shared semantic space of dimension $d = 512$ using a linear layer included in the

trainable parameters ϕ_g. The intuition behind this approach is to ensure that semantically similar proofs (those with similar DAG structures of proof-states and tactics) produce similar embeddings.

Contrastive Learning. To enable cross-modal retrieval between NL and FL representations, it is essential to align the two modalities in the embedding space. Specifically, for each positive pair $\left(M_{\text{NL}}^{(i)}, M_{\text{FL}}^{(i)}\right)$, we aim for their embeddings $\left(v_{\text{NL}}^{(i)}, v_{\text{FL}}^{(i)}\right)$ to exhibit high cosine similarity, while embeddings of mismatched pairs are pushed apart. Denoting ℓ_2-normalization by $\widehat{v} = v/\|v\|_2$ and defining the cosine similarity between two embeddings u and w as $[\widehat{u}, \widehat{w}]$, we adopt the following symmetric contrastive loss for a mini-batch $\mathcal{B} = \left\{\left(M_{\text{NL}}^{(i)}, M_{\text{FL}}^{(i)}\right)\right\}_{i=1}^{n}$ of NL-FL pairs:

$$\mathcal{L}(\mathcal{B}) = -\frac{1}{2n}\sum_{i=1}^{n}\left[\log\left(\frac{\exp\left(\left[\widehat{v}_{\text{NL}}^{(i)}, \widehat{v}_{\text{FL}}^{(i)}\right]/\tau\right)}{\sum_{j=1}^{n}\exp\left(\left[\widehat{v}_{\text{NL}}^{(i)}, \widehat{v}_{\text{FL}}^{(j)}\right]/\tau\right)}\right) + \log\left(\frac{\exp\left(\left[\widehat{v}_{\text{FL}}^{(i)}, \widehat{v}_{\text{NL}}^{(i)}\right]/\tau\right)}{\sum_{j=1}^{n}\exp\left(\left[\widehat{v}_{\text{FL}}^{(i)}, \widehat{v}_{\text{NL}}^{(j)}\right]/\tau\right)}\right)\right] \quad (1)$$

where $\tau > 0$ is a temperature hyperparameter. This loss encourages each NL embedding to be closest to its corresponding FL embedding, and vice versa, using other in-batch embeddings as negatives. The negatives are sampled randomly for each mini-batch.

NL/FL Cross-Modal Retrieval. We precompute the normalized embeddings $\left\{\widehat{v}_{\text{NL}}^{(i)}\right\}_{i=1}^{|\mathcal{D}|}$ and $\left\{\widehat{v}_{\text{FL}}^{(j)}\right\}_{j=1}^{|\mathcal{D}|}$ for all NL and FL theorem–proof pairs respectively in our database $\mathcal{D}$, which enables efficient cross-modal retrieval. Given a query theorem–proof pair in either source modality (NL or FL), we encode it into the shared semantic space (yielding $\widehat{q}_{\text{NL}}$ or $\widehat{q}_{\text{FL}}$) and compute cosine similarities with all items in the target modality, producing the set $\left\{\left[\widehat{q}_{\text{NL}}, \widehat{v}_{\text{FL}}^{(j)}\right]\right\}_{j=1}^{|\mathcal{D}|}$ or $\left\{\left[\widehat{q}_{\text{FL}}, \widehat{v}_{\text{NL}}^{(i)}\right]\right\}_{i=1}^{|\mathcal{D}|}$, depending on the retrieval direction. The top-K nearest neighbors from these sets are then selected as demonstrations, reflecting similar proof structures, patterns, and mathematical domains.

4.2 Retrieval-Augmented Fine-Tuning for Proof Auto-Formalization

We fine-tune an LLM to translate NL theorem-proof pairs into FL (Lean), conditioned on retrieved FL demonstrations that provide rich contextual demonstrations. For each training instance, we construct a prompt containing (a) input NL theorem-proof pair M_{NL} and (b) top-K retrieved FL theorem-proof pairs: $\mathcal{R}(M_{\text{NL}}, \mathcal{D}) = \{M_{\text{FL}}^{(k)}\}_{k=1}^{K}$ with relevance scores $\{r^{(k)}\}_{k=1}^{K}$. The retrieved examples demonstrate how similar mathematical concepts and proof strategies are formalized in Lean. We include relevance scores to help the model weight the importance of each retrieved example.

Training Objective. We fine-tune Kimina-Prover-RL-1.7B (Wang et al., 2025) using supervised learning on our NuminaMath-Lean-PF dataset. The model is trained to generate an FL theorem-proof pair $\widetilde{M}_{\text{FL}}$ given the input context. This retrieval-augmented approach allows the LLM to learn from similar formalization patterns rather than generating formal theorems in isolation. As illustrated in Figure 1b, we use the standard auto-regressive language modeling loss:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{|\mathcal{T}|}\sum_{t=1}^{|\mathcal{T}|}\log P_\theta\left(\tau_t \mid \tau_{<t}, \mathcal{C}\right) \quad (2)$$

where $\mathcal{T} = \widetilde{M}_{\text{FL}}$ is the generated formalization tokenized as sequence $(\tau_1, \ldots, \tau_{|\mathcal{T}|})$, $\mathcal{C}$ represents the input context (NL theorem-proof + retrieved FL examples), and θ are the LLM parameters. This corresponds to the cross-entropy loss between $\widetilde{M}_{\text{FL}}$ and the gold-standard formalization M_{FL}.

4.3 Iterative Proof Repair with Verifier Feedback

During inference, we perform retrieval-augmented proof auto-formalization with the fine-tuned LLM (Figure 1c). However, LLM being a stochastic model may still generate FL theorem-proof pair that contain syntactic errors or semantic misalignments with the input NL theorem-proof. To address, we implement an iterative repair mechanism that combines Lean's type checker with semantic equivalence verification. For an input NL theorem-proof pair $M_{\text{NL}} = \langle T_{\text{NL}}, P_{\text{NL}}\rangle$ the LLM generates an FL counterpart $\widetilde{M}_{\text{FL}} = \langle \widetilde{T}_{\text{FL}}, \widetilde{P}_{\text{FL}}\rangle$, on which we perform two types of verification:

1. **Syntactic Verification:** We compile $\widetilde{M}_{\text{FL}}$ using Lean's type checker. If compilation fails, we extract the specific error message and location from Lean's diagnostic output.

2. **Semantic Verification:** We assess whether the generated theorem $\widetilde{T}_{\text{FL}}$ accurately represents the original NL theorem T_{NL} using an LLM-based equivalence judge.

Repair Process. When either syntactic or semantic verification fails, we initiate an iterative repair process. The procedure terminates once both checks succeed or the maximum number of repair attempts ($R_{\max} = 5$) is reached. This bounded, iterative strategy improves the reliability of proof auto-formalization by catching and correcting common errors while maintaining computational efficiency. The overall process is described in Algorithm 1.

Algorithm 1 Iterative Proof Repair

Input: NL theorem-proof pair $M_{\text{NL}} = \langle T_{\text{NL}}, P_{\text{NL}} \rangle$,
Initial FL theorem-proof pair $\widetilde{M}_{\text{FL}}^{(0)} = \langle \widetilde{T}_{\text{FL}}^{(0)}, \widetilde{P}_{\text{FL}}^{(0)} \rangle$
Output: Verified FL theorem-proof pair or FAILURE
for $i = 0$ to $R_{\max} - 1$ **do**
  syntaxOK $\leftarrow$ LeanTypeCheck$\big(\widetilde{M}_{\text{FL}}^{(i)}\big)$
  semanticsOK $\leftarrow$ SemanticEquivalence$\big(T_{\text{NL}}, \widetilde{T}_{\text{FL}}^{(i)}\big)$
  if syntaxOK $\wedge$ semanticsOK **then return** $\widetilde{M}_{\text{FL}}^{(i)}$
  end if
  feedback $\leftarrow$ GenerateFeedback(syntaxOK, semanticsOK)
  $\widetilde{M}_{\text{FL}}^{(i+1)} \leftarrow$ LLMRepair(feedback)
end for
return FAILURE

5 Experimental Evaluation

5.1 Data Preparation: NuminaMath-Lean-PF and miniF2F-Test-PF

For validation, we curate miniF2F-Test-PF by combining two versions of miniF2F-test (Zheng et al., 2021), a widely-used auto-formalization benchmark, using the Lean v4.15.0 version (NuminaMath, 2025) and obtaining missing NL proofs from Yang (2025). For training, we construct NuminaMath-Lean-PF from NuminaMath-LEAN (Wang et al., 2025), containing 104,155 competition-level problems in algebra, geometry, number theory, combinatorics, and calculus. Each instance pairs an NL theorem T_{NL} with a human-written Lean v4.15.0 theorem T_{FL}; 38,951 include FL proofs P_{FL} (30% human-written, rest by KiminaProver), forming $\{T_{\text{NL}}, \langle T_{\text{FL}}, P_{\text{FL}} \rangle\}$. Next, we prepare NuminaMath-Lean-PF via the following steps.

Formal Verification and Repair. Each FL pair is type-checked in Lean (Leanprover, 2025). About 6% (2,337) failed due to syntax, library mismatches, or incomplete proofs. These were automatically repaired via Gemini-2.5-Pro: error messages and locations are extracted from Lean, used to prompt the LLM for corrections, and re-verified iteratively up to five times. All errors were successfully fixed, showing most issues were syntactic rather than mathematical.

NL Proof Generation. NuminaMath-LEAN provides only theorems, so we generate NL proofs in two stages. First, *solution sketch retrieval* matches T_{NL} to NuminaMath 1.5 (Li et al., 2024) (896k problem-solution pairs) via exact string matching, retrieving sketches (median 79 words) for 25,792 instances (66%). Second, *FL-to-NL informalization* uses Gemini-2.5-Pro to translate verified P_{FL} into detailed, human-readable NL proofs. Each instance uses P_{FL}, $\langle T_{\text{FL}}, T_{\text{NL}} \rangle$, and sketches when available, producing 38,951 NL–FL theorem–proof pairs $\{\langle T_{\text{NL}}, P_{\text{NL}} \rangle, \langle T_{\text{FL}}, P_{\text{FL}} \rangle\}$.

5.2 Experimental Setup, Evaluation Metrics and State-of-the-Arts

We train our joint embedding model on 90% (35,056 instances) of NuminaMath-Lean-PF and evaluate it on the remaining 3,895 instances. For subsequent steps, we treat the train split as a database $\mathcal{D}$ of FL theorem-proof pairs. The LLM is SFT-tuned for NL-to-FL translation using $\big(M_{\text{NL}}, M_{\text{FL}}\big)$ from NuminaMath-Lean-PF, with the joint embedding model retrieving the top-5 relevant FL pairs from $\mathcal{D}$ for each input M_{NL}. The resulting LLM (ProofBridge) is evaluated on miniF2F-Test-PF for proof auto-formalization, again using $\mathcal{D}$ for cross-modal retrieval.

Metrics for NL/FL Cross-Modal Retrieval. We evaluate cross-modal alignment of our joint embedding model in two directions. NL $\rightarrow$ FL measures retrieval of FL theorem-proof pairs given an NL input, which is relevant for proof auto-formalization, while FL $\rightarrow$ NL assesses the reverse. For a test pair $\big(M_{\text{NL}}, M_{\text{FL}}\big)$, a retrieval in the NL $\rightarrow$ FL direction is deemed successful if the model retrieves the FL counterpart M_{FL} given M_{NL}, and unsuccessful otherwise; the FL $\rightarrow$ NL direction is evaluated analogously. We assess retrieval performance using four metrics. ***(i) Recall Rate @ K*** measures the percentage of queries for which the query's cross-modal counterpart appears among the top-K retrieved results. We report $K = 1, 5, 10, 20, 50$. ***(ii) Mean Reciprocal Rank (MRR)*** is the average reciprocal rank of the retrieved cross-modal counterpart for each query, $\text{MRR} = \frac{1}{N} \sum_{q=1}^{N} \frac{1}{\text{rank}_q}$, indicating how highly it is ranked. ***(iii) Cosine Similarity of Top-K Retrieved*** measures the cosine similarity between the query embedding and those of the top-K re-

trieved instances. For each query, we sort these scores in ascending order and record three statistics: median (**M**), 25^{th} percentile (**Q1**), and 75^{th} percentile (**Q3**), and report their average over the test set. ***(iv) Cosine Similarity of Non-Retrieved*** applies the same procedure to all non-retrieved instances and reports the median (**M**), 25^{th} percentile (**Q1**), and 75^{th} percentile (**Q3**) averaged over the test set.

Metrics for Proof Auto-Formalization. Given $M_{\text{NL}} = \langle T_{\text{NL}}, P_{\text{NL}} \rangle$, an auto-formalizer LLM/tool generates an FL version $\widetilde{M}_{\text{FL}} = \langle \widetilde{T}_{\text{FL}}, \widetilde{P}_{\text{FL}} \rangle$. We evaluate performance using two metrics: ***(i) Type Correctness (TC)*** measures whether $\widetilde{M}_{\text{FL}}$ is accepted by Lean's type-checker, i.e., $\widetilde{P}_{\text{FL}}$ proves $\widetilde{T}_{\text{FL}}$ without using `sorry`. ***(ii) Semantic Correctness (SC)*** is evaluated only for type-correct generations and measures whether $\widetilde{T}_{\text{FL}}$ is definitionally equal [1] to the gold-standard T_{FL} by prompting Gemini 2.5 Pro up to five times to produce a Lean proof of the bi-directional equivalence $\widetilde{T}_{\text{FL}} \leftrightarrow T_{\text{FL}}$ using a restricted set of tactics (`rfl`, `simp`, `ring`, etc.; details in Appendix A.3). Although relying on an LLM judge, it is based on the Lean proof of equivalence rather than the LLM's judgment alone. TC and SC are computed as *pass@k*, i.e., over the top-k generated candidates, for $k = 1, 2, 4, 8, 16, 32$.

SoTA for NL/FL Cross-Modal Retrieval. To our knowledge, no existing model jointly embeds theorems and proofs in NL and FL. Constructing such a shared semantic space is non-trivial, as pre-trained encoders alone cannot produce embeddings that support meaningful cross-modal retrieval. To illustrate, we introduce *SoTA Encoder* as a comparison: it treats both NL and FL theorem–proof pairs as plain text, encodes them using the SoTA `all-MiniLM-L6-v2` (Reimers & Gurevych, 2020), and performs retrieval via cosine similarity in this embedding space.

SoTA Tools for Proof Auto-Formalization. Existing proof auto-formalization tools include DSP (Jiang et al., 2022) which only supports Isabelle while we only support Lean 4, and hence we cannot compare against it. Another recent tool is FormL4 (Lu et al., 2024), which unfortunately has not been released. Hence, we compare against foundation models, SoTA automated FL proof synthesis (AFPS), and NL-to-FL theorem auto-formalization tools. The four foundation models we include in our experiments are: GPT-5-mini (OpenAI, 2025) and the Gemini-2.5 variants Flash-Lite, Flash, and Pro. The AFPS models we include are DeepSeek-Prover-V1.5-RL (Xin et al., 2024b), STP_model_Lean_0320 (Dong & Ma, 2025), Goedel-Prover-SFT (Lin et al., 2025), and Leanabell-Prover-V2-KM (Zhang et al., 2025). We focus on AFPS LLMs because they share the same output space as proof auto-formalization, i.e., full FL proofs with tactics. The SoTA auto-formalization tools we use in our experiments are Kimina-Autoformalizer-7B (Wang et al., 2025) and Herald-Translator (Gao et al., 2024); unfortunately, they only auto-formalize theorem statements and not proofs (i.e., produce `sorry` for proofs), yielding 0% TC and 0% SC in our setting. All models are evaluated in the few-shot (three-example) setting, with the exception of Kimina-Prover-RL-1.7B (Wang et al., 2025), our base model, which is tested in both few-shot and zero-shot settings.

5.3 Experimental Results and Ablation Studies

NL/FL Cross-Modal Retrieval. Table 1 compares ProofBridge's joint embedding with the *SoTA Encoder*, showing that ProofBridge achieves higher Recall Rates across all top-K values. For NL→FL, it yields up to a $3.28\times$ gain at pass@1, while for FL→NL the gain is up to $1.94\times$ at pass@1. MRR also improves by $2.74\times$ and $1.79\times$ for NL→FL and FL→NL, respectively. Furthermore, ProofBridge typically retrieves the correct cross-modal counterpart within the top-2, with a median rank of 1, confirming that it consistently appears among the top retrieved results. For the joint embeddings learned by ProofBridge, the median (**M**) cosine similarity shows that select cross-modal theorem-proof pairs are closely embedded, with top-K retrieved instances ($K = 1, 5, 10, 20, 50$) averaging 0.64 in both NL→FL and FL→NL directions. In contrast, most non-retrieved instances are placed much farther apart, averaging -0.01. Compared to the *SoTA Encoder*, these values are roughly 23% higher for retrieved instances and 104% lower for non-retrieved ones. This indicates that ProofBridge keeps unrelated theorem–proof pairs far apart while embedding the correct cross-modal counterparts close together. Overall, our results show that combining contrastive learning with extracting and encoding linear DAG traversals of Lean proofs yields a significantly more effective model for constructing a joint embedding space for NL and FL theorem–proof pairs. Consequently, semantically equivalent NL–FL pairs appear close in the joint space, whereas inequivalent pairs remain distant, enabling reliable retrieval of relevant FL theorem–proof pairs to condition the LLM for proof auto-formalization.

[1] We admit some propositional equalities in addition to definitional ones, see details in Appendix A.3

Table 1: **NL/FL Cross-Modal Retrieval Performance.** Comparison of *SoTA Encoder* and PROOFBRIDGE's joint embedding across retrieval metrics. (`Q1` = 25^{th} %tile, `M` = median, `Q3` = 75^{th} %tile)

Method	Retrieval Direction	Recall Rate @ K (%) ↑					MRR ↑		Cos. Similarity of top-K Retrieved ↑					Cos. Similarity of NOT Retrieved ↓				
		K=1	K=5	K=10	K=20	K=50			K=1	K=5	K=10	K=20	K=50	K=1	K=5	K=10	K=20	K=50
all-MiniLM-L6-v2 (SoTA Encoder)	NL → FL	16.06	30.93	38.63	47.31	60.95	0.237	`Q1`:	0.60	0.52	0.50	0.47	0.44	0.147	0.147	0.147	0.147	0.146
								`M`:	0.60	0.54	0.51	0.49	0.45	0.210	0.210	0.210	0.209	0.208
								`Q3`:	0.60	0.55	0.53	0.51	0.58	0.274	0.274	0.273	0.273	0.271
	FL → NL	26.35	45.12	54.28	63.75	75.54	0.355	`Q1`:	0.58	0.52	0.50	0.47	0.44	0.142	0.142	0.142	0.142	0.141
								`M`:	0.58	0.53	0.51	0.49	0.46	0.208	0.208	0.208	0.207	0.206
								`Q3`:	0.58	0.54	0.53	0.51	0.48	0.278	0.278	0.277	0.277	0.275
PROOFBRIDGE	NL → FL	**52.83**	**79.81**	**87.06**	**91.49**	**95.08**	**0.650**	`Q1`:	**0.76**	**0.66**	**0.62**	**0.57**	**0.49**	**-0.134**	**-0.135**	**-0.135**	**-0.135**	**-0.136**
								`M`:	**0.76**	**0.68**	**0.64**	**0.60**	**0.52**	**-0.009**	**-0.010**	**-0.010**	**-0.010**	**-0.012**
								`Q3`:	**0.76**	**0.71**	**0.68**	**0.64**	**0.58**	**0.123**	**0.123**	**0.122**	**0.121**	**0.117**
	FL → NL	**51.23**	**78.77**	**86.18**	**90.50**	**94.83**	**0.635**	`Q1`:	**0.77**	**0.67**	**0.62**	**0.57**	**0.49**	**-0.135**	**-0.135**	**-0.135**	**-0.135**	**-0.136**
								`M`:	**0.77**	**0.69**	**0.64**	**0.60**	**0.52**	**-0.009**	**-0.009**	**-0.009**	**-0.011**	**-0.012**
								`Q3`:	**0.77**	**0.71**	**0.68**	**0.64**	**0.58**	**0.124**	**0.123**	**0.122**	**0.121**	**0.117**

Table 2: **Proof Auto-Formalization Performance.** Comparison of LLM-based tools on *Semantic Correctness (SC)* and *Type Correctness (TC)* across pass@k metrics ($k \in \{1, 2, 4, 8, 16, 32\}$).

LLM/Tool	Semantic Correctness (SC) (%) ↑						Type Correctness (TC) (%) ↑					
	pass@1	pass@2	pass@4	pass@8	pass@16	pass@32	pass@1	pass@2	pass@4	pass@8	pass@16	pass@32
Kimina-Autoformalizer-7B (few-shot)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Herald-translator (few-shot)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Gemini-2.5-Flash-Lite (few-shot)	0.00	0.00	0.00	0.00	1.23	2.87	0.82	4.51	9.02	13.93	18.03	21.31
Gemini-2.5-Flash (few-shot)	0.00	0.82	2.05	2.86	3.28	4.10	2.45	4.92	7.38	12.30	15.98	18.44
Gemini-2.5-Pro (few-shot)	1.23	1.23	3.28	4.92	6.97	8.61	9.84	13.52	18.85	23.77	29.10	31.56
GPT-5-mini (few-shot)	0.41	1.23	4.51	6.97	7.38	9.02	4.92	9.43	20.08	28.28	32.38	34.84
DeepSeek-Prover-V1.5-RL (few-shot)	3.69	6.15	8.61	9.02	11.07	12.30	8.20	14.34	19.67	24.18	28.68	35.66
STP_model_Lean_0320 (few-shot)	4.51	6.56	8.20	9.84	11.48	13.11	12.70	18.03	23.36	28.69	33.61	39.34
Goedel-Prover-SFT (few-shot)	4.92	5.33	7.38	8.20	12.70	16.39	13.52	17.21	25.41	31.56	36.88	42.21
Leanabell-Prover-V2-KM (few-shot)	6.97	9.43	10.66	13.52	15.16	18.03	16.80	21.31	27.87	37.30	41.39	50.41
Kimina-Prover-RL-1.7B (zero-shot)	9.02	13.93	22.54	30.33	35.25	40.16	26.23	41.80	56.15	62.70	68.03	75.00
Kimina-Prover-RL-1.7B (few-shot)	6.15	12.30	17.62	22.13	27.46	31.56	26.23	42.21	60.66	74.18	88.11	93.85
PROOFBRIDGE (SFT only)	6.97	13.52	19.26	24.18	29.92	34.84	27.87	45.90	60.66	66.39	72.13	78.69
PROOFBRIDGE (Retrieval-augmt. SFT)	13.11	20.90	27.87	35.66	47.95	55.33	29.92	46.31	60.25	71.31	83.20	89.75
PROOFBRIDGE (Retrieval-augmt. SFT + Repair)	**16.39**	**25.41**	**29.51**	**37.70**	**50.41**	**62.70**	**32.79**	**47.13**	**64.75**	**78.69**	**90.16**	**95.49**

Proof Auto-Formalization. As shown in Table 2, theorem auto-formalization models perform poorly, with 0% correctness across all pass@k. These tools lack knowledge of proof DAGs and tactics and output only FL theorems, with proofs as `sorry`. Among foundation models, Gemini-2.5-Flash-Lite achieves 2.87% SC and 21.31% TC at pass@32, which increase to 4.10% SC, 18.44% TC for Flash and 8.61% SC, 31.56% TC for Pro. GPT-5-mini attains 9.02% TC and 34.84% SC at pass@32. Among the SoTA AFPS LLMs, Kimina-Prover-RL-1.7B performs best in the few-shot setting, achieving 31.56% SC and 93.85% TC at pass@32. In zero-shot, SC rises to 40.16% while TC drops to 75.00%. This indicates that while random in-context examples can improve TC (syntax) but degrade SC (semantics), leading the model to hallucinate proofs that do not align with the input NL. This underscores the importance of providing semantically relevant examples, as implemented in PROOFBRIDGE. We first evaluate **PROOFBRIDGE (SFT)**, trained solely on labeled NL–FL theorem-proof pairs and evaluated in the few-shot setting, improving SC by +3.28% but reducing TC by -15.16% compared to Kimina-Prover-RL-1.7B (few-shot). In **PROOFBRIDGE (Retrieval-augmented SFT)**, fine-tuned and inferred with five relevant FL proofs retrieved via cross-modal retrieval, SC rises by +23.77% and TC drops by -4.1% relative to Kimina-Prover-RL-1.7B (few-shot). Finally, **PROOFBRIDGE (Retrieval-augmented SFT + Repair)**, which incorporates iterative proof repair, achieves +31.14% SC and +1.64% TC over Kimina-Prover-RL-1.7B (few-shot).

6 CONCLUSION

We present PROOFBRIDGE, a unified framework for NL-to-Lean proof auto-formalization that translates both theorems and proofs end-to-end[2]. At its core is a joint embedding model of NL and FL that encodes Lean proof DAGs, capturing tactic sequences and dependency structures. It enables highly effective cross-modal retrieval of semantically relevant FL proofs. These retrieved proofs act as demonstrations, guiding retrieval-augmented fine-tuning of an LLM. An iterative verifier-guided repair loop further refines generated proofs by combining Lean type-checking with semantic equivalence checking to ensure correctness. Evaluated on MINIF2F-TEST-PF, PROOFBRIDGE significantly outperforms state-of-the-art LLMs in both semantic correctness (by bi-directional equivalence proving) and type correctness, demonstrating that integrating structured embeddings, retrieval guidance, and verifier feedback leads to more reliable proof auto-formalization.

[2]**Reproducibility:** System details (Appendix A.2), code and datasets provided in the supplementary.

REFERENCES

Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Yann Coscoy, David Delahaye, Daniel de Rauglaudre, Jean-Christophe Filliâtre, Eduardo Giménez, Hugo Herbelin, et al. The Coq Proof Assistant Reference Manual. *INRIA, version*, 6(11), 1999.

Kevin Buzzard. Formalising mathematics. Lecture notes from a course at Imperial College London, 2022. URL https://github.com/ImperialCollegeLondon/formalising-mathematics.

Google Deepmind. AI achieves Silver-Medal Standard solving International Mathematical Olympiad Problems, 2024. URL https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/. Accessed: Sep, 2025.

Kefan Dong and Tengyu Ma. Beyond Limited Data: Self-play LLM Theorem Provers with Iterative Conjecturing and Proving. *arXiv preprint arXiv:2502.00212*, 2025.

Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A Natural Language Annotated Lean 4 Dataset. *arXiv preprint arXiv:2410.10878*, 2024.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward Ayers, and Stanislas Polu. Proof Artifact Co-Training for Theorem Proving with Language Models. In *International Conference on Learning Representations*, 2022.

John Harrison. HOL Light: An Overview. In *International Conference on Theorem Proving in Higher Order Logics*, pp. 60–66. Springer, 2009.

Prithwish Jana. NeuroSymbolic LLM for mathematical reasoning and software engineering. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 8492–8493, 2024.

Prithwish Jana, Piyush Jha, Haoyang Ju, Gautham Kishore, Aryan Mahajan, and Vijay Ganesh. CoTran: An LLM-based Code Translator using Reinforcement Learning with Feedback from Compiler and Symbolic Execution. In *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI)*, pp. 4011–4018, 2024.

Piyush Jha, Prithwish Jana, Pranavkrishna Suresh, Arnav Arora, and Vijay Ganesh. RLSF: Reinforcement Learning via Symbolic Feedback. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI)*, 2025.

Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs. *arXiv preprint arXiv:2210.12283*, 2022.

Albert Q Jiang, Wenda Li, and Mateja Jamnik. Multi-language Diversity Benefits Autoformalization. *Advances in Neural Information Processing Systems*, 37:83600–83626, 2024.

Albert Qiaochu Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. LISA: Language models of ISAbelle proofs. In *6th Conference on Artificial Intelligence and Theorem Proving*, pp. 378–392, 2021.

Leanprover. leanprover-community/repl: A simple repl for lean 4, 2025. URL https://github.com/leanprover-community/repl. Accessed: Sep, 2025.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath. [https://huggingface.co/AI-MO/NuminaMath-1.5](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, et al. Goedel-Prover: A Frontier Model for Open-Source Automated Theorem Proving. *arXiv preprint arXiv:2502.07640*, 2025.

Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen, Hui Jin, Jipeng Zhang, Haiming Wang, et al. Process-driven Autoformalization in Lean 4. *arXiv preprint arXiv:2406.01940*, 2024.

Norman Megill and David A Wheeler. *Metamath: A Computer Language for Mathematical Proofs*. Lulu. com, 2019.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pp. 625–635. Springer, 2021.

Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer, 2002.

NuminaMath. minif2f_test. Hugging Face Dataset, September.23 2025. URL https://huggingface.co/datasets/AI-MO/minif2f_test. Version 1.0, Apache-2.0 License, 244 rows.

OpenAI. Introducing gpt-5, 2025. URL https://openai.com/index/introducing-gpt-5/. Accessed: 2025-09-23.

Lawrence C Paulsson and Jasmin C Blanchette. Three Years of Experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers. In *Proceedings of the 8th International Workshop on the Implementation of Logics (IWIL-2010), Yogyakarta, Indonesia. EPiC*, volume 2, 2012.

Gabriel Poesia and Noah D Goodman. Peano: Learning Formal Mathematical Reasoning. *Philosophical Transactions of the Royal Society A*, 381(2251):20220044, 2023.

Stanislas Polu and Ilya Sutskever. Generative Language Modeling for Automated Theorem Proving. *arXiv preprint arXiv:2009.03393*, 2020.

Nils Reimers and Iryna Gurevych. Sentence-Transformers: Multilingual Sentence Embeddings using BERT and XLNet. https://www.sbert.net/, 2020.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-Prover-v2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Terence Tao. A maclaurin type inequality. *arXiv preprint arXiv:2310.05328*, 2023.

Amitayush Thakur, Yeming Wen, and Swarat Chaudhuri. A Language-Agent Approach to Formal Theorem-Proving. 2023.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. SynCode: LLM Generation with Grammar Augmentation. *arXiv preprint arXiv:2403.01632*, 2024.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-Prover Preview: Towards Large Formal Reasoning Models with Reinforcement Learning. *arXiv preprint arXiv:2504.11354*, 2025.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. Theorem-Llama: Transforming General-purpose LLMs into Lean4 Experts. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 11953–11974, 2024.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with Large Language Models. *Advances in Neural Information Processing Systems*, 35:32353–32368, 2022.

Yutong Wu, Di Huang, Ruosi Wan, Yue Peng, Shijie Shang, Chenrui Cao, Lei Qi, Rui Zhang, Zidong Du, Jie Yan, et al. Stepfun-formalizer: Unlocking the autoformalization potential of llms through knowledge-reasoning fusion. *arXiv preprint arXiv:2508.04440*, 2025.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. DeepSeek-Prover: Advancing Theorem Proving in LLMs through Large-Scale Synthetic Data. *arXiv preprint arXiv:2405.14333*, 2024a.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024b.

Kaiyu Yang. minif2f-lean4. GitHub repository, 2025. URL `https://github.com/yangky11/miniF2F-lean4`.

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.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean Workbook: A Large-scale Lean Problem Set Formalized from Natural Language Math Problems. *Advances in Neural Information Processing Systems*, 37:105848–105863, 2024.

Jingyuan Zhang, Qi Wang, Xingguang Ji, Yahui Liu, Yang Yue, Fuzheng Zhang, Di Zhang, Guorui Zhou, and Kun Gai. Leanabell-prover: Posttraining scaling in formal reasoning. *arXiv preprint arXiv:2504.06122*, 2025.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. MiniF2F: A Cross-System Benchmark for Formal Olympiad-level Mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

Yuhao Zhou. Retrieval-Augmented TLAPS Proof Generation with Large Language Models. *arXiv preprint arXiv:2501.03073*, 2025.

A APPENDIX

A.1 THE USE OF LARGE LANGUAGE MODELS (LLMS)

LLMs did *not* play a significant role in either the research ideation or the writing of this paper. Their use was limited to correcting minor grammatical issues and typographical errors.

A.2 REPRODUCIBILITY

All implementations in this work, including dataset construction, model training, cross-modal retrieval, inference, and evaluation, use Python 3.12.10 and Lean v4.15.0. Experiments were conducted on a high-performance AlmaLinux 9.5 (Teal Serval) cluster with a single Intel Xeon Platinum 8480+ CPU (32 cores, 2.0–4.0 GHz), 251 GiB of RAM, and one NVIDIA H100 GPU. Our full codebase, including scripts for dataset generation, model fine-tuning, and inference across both GPU- and API-based setups, is shared through a supplementary .zip file. The repository is complete, well-documented, and designed for full reproducibility: it includes instructions for creating a Python virtual environment, and a comprehensive README outlining library dependencies, dataset format, and step-by-step instructions to replicate the entire data pipeline and experimental workflow.

A.3 SEMANTIC EQUIVALENCE OF LEAN THEOREMS

The task of autoformalization is to convert a mathematical theorem and proof from natural language into a formal language, such as Lean. When evaluating the performance of such systems, we propose two criteria for evaluation: *type correctness* and *semantic equivalence*. Type correctness, which requires that the generated Lean proof is accepted by the Lean type-checker, is straightforward to verify and serves as the standard evaluation metric in the field. However, semantic equivalence, ensuring the FL theorem faithfully represents the meaning of the original NL theorem, presents a far greater challenge. To the best of our knowledge, semantic equivalence has not been systematically evaluated in prior work. This section introduces a novel methodology towards addressing this gap.

While directly measuring the semantic alignment between a NL theorem and a Lean theorem is an unsolved challenge, showing the logical equivalence of two Lean theorems is a tractable task. Our training dataset, NUMINAMATH-LEAN-PF, contains pairs of $\langle T_{\text{NL}}, T_{\text{FL}} \rangle$ where most of the T_{FL} were manually created by experts at Numina. We treat these high-quality T_{FL} theorems as gold-standard references, assuming they are faithful translations of their T_{NL} counterparts. This allows us to reduce the intractable problem of verifying a model's generated theorem $\widetilde{T}_{\text{FL}}$ against the original T_{NL} to the more tractable task of checking for logical equivalence between $\widetilde{T}_{\text{FL}}$ and the golden reference T_{FL}, which can be checked in Lean itself.

To be more specific, we enforce this semantic equivalence check by proving the logical biconditional $\widetilde{T}_{\text{FL}} \leftrightarrow T_{\text{FL}}$ in Lean. Theorems like $\widetilde{T}_{\text{FL}}$ and T_{FL} are of type `Prop` in Lean. The following theorem from Mathlib states that for any two propositions, a logical biconditional between two propositions is itself logically equivalent to their propositional equality:

```
theorem propext_iff{a b : Prop} :
a = b ↔ (a ↔ b)
```

The task thus converts to proving the equality $\widetilde{T}_{\text{FL}} = T_{\text{FL}}$ within Lean. This requires clarifying the specific notion of equality being used, as Lean distinguishes between three primary types: *syntactic*, *definitional*, and *propositional* Buzzard (2022). Syntactic equality is the strictest form of equality in Lean, as it only admits expressions that are structurally identical according to their Abstract Syntax Trees, without any computation or reduction. Definitional equality is a more relaxed form of equality than syntactic equality, where two expressions are considered equal if they compute or reduce to the same normal form. Propositional equality is the weakest form of equality, and also the standard notion of equality used in mathematical theorems. Two terms `a, b` are propositionally equal in Lean if you can construct a proof term for the proposition `a = b`.

For our evaluation, we seek to measure how closely a $\widetilde{T}_{\text{FL}}$ matches the T_{FL}. The strictest criterion, syntactic equality, is too restrictive given the current state-of-the-art, as it would fail valid theorems

with trivial notational differences. Conversely, full propositional equality can be too permissive; a proof of equivalence can be arbitrarily complex, making it difficult to automate and decide.

Therefore, we adopt a pragmatic compromise: we check for **definitional equality** supplemented by a form of **bounded propositional equality**. This means we primarily check if $\widetilde{T}_{\text{FL}}$ and T_{FL} reduce to the same normal form, but we also permit some propositional equality, provided they can be proven using a collection of tactics so that their proof complexity is bounded.

We then leverage Gemini 2.5 Pro as an automated equivalence checker. The model is prompted to synthesize a proof for the biconditional theorem ($\widetilde{T}_{\text{FL}} \leftrightarrow T_{\text{FL}}$), with instructions limiting it to a specific subset of available tactics. This restricted set includes three powerful automated tactics, `rfl`, `simp`, and `ring`, each is designed to discharge a specific class of goals: `rfl` for definitional equality, `simp` for simplification, and `ring` for polynomial identities.

As definitional equality is our primary target, the equivalence checker first attempts to solve the goal with the `rfl` tactic. This single tactic should suffice for the majority of cases. If `rfl` fails, the checker then tries `simp`. This tactic performs additional simplifications by rewriting the goal using theorems from Mathlib that are tagged for its use. Critically, we use `simp` without any arguments. Providing explicit arguments would require a demanding search for the correct lemmas and could introduce unbounded complexity, violating our goal of a bounded proof search. Furthermore, the need for `simp` with arguments could imply that the required rewrite is non-trivial, since the default simplification set contains most of the trivial facts [3]. Since our goal is to ensure a close correspondence between $\widetilde{T}_{\text{FL}}$ and T_{FL}, a proof requiring such a targeted rewrite indicates a semantic distance that we classify as a mismatch. The `ring` tactic is a valuable complement to the previous tactics as it specializes in proving polynomial equalities. The `ring` tactic operates by reducing arithmetic expressions to a canonical normal form. This allows it to prove the equivalence of expressions that are algebraically identical but not definitionally so, such as `x^2` and `x * x`, which `rfl` and default `simp` would otherwise fail to solve.

The three tactics discussed above cover most of the direct equivalences we aim to check. The remaining tactics in our instruction set are designed for a more nuanced case: proving the biconditional when two theorems differ only in their use of auxiliary variables. We observed that human experts and language models may make different but equally valid decisions on whether to introduce an auxiliary variable. We therefore classify such theorems as equivalent. For example, consider the following:

```
def Prop1 := (∀ (b h v : ℝ), (0 < b ∧ 0 < h ∧ 0 < v) → (v = 1 / 3 * (b *
     h)) → (b = 30) → (h = 13 / 2) → v = 65)
def Prop2 := (∀ {B h : ℝ}, (B = 30) → (h = 6.5) → (1 / 3) * B * h = 65)
example : Prop1 ↔ Prop2 := by
  constructor
  · intro
    simp
  · ring
    simp
    intros
    nlinarith
```

The main difference between the two propositions `Prop1` and `Prop2` is the presence of the auxiliary variable `v` in one. To prove that such theorems are equivalent, one must typically prove the biconditional by separately proving the implications of both direction. This requires a step-by-step proof construction, and the tactics above are included in our instruction set.

[3] It is important to note that the default simp set intentionally excludes lemmas like associativity and commutativity, as they can cause the simplifier to loop indefinitely. However, since these lemmas primarily concern algebraic expressions, they can be handled by the `ring` tactic.