HybridProver: Augmenting Theorem Proving with LLM-Driven Proof Synthesis and Refinement

Anonymous ACL submission

Abstract

Formal methods play a crucial role in ensuring the reliability of critical systems through rigorous mathematical verification. However, their adoption remains limited due to the labor-intensive nature of manual proof construction. Recent advances in large language models (LLMs) have opened new opportunities for automated theorem proving. Two main paradigms have emerged: stepwise tactic-based generation and whole-proof synthesis. While both approaches have complementary strengths, existing work largely treats them in isolation. In this work, we propose *HybridProver*, a unified framework that integrates whole-proof synthesis and tactic-based generation through *proof sketches* as an intermediate representation. This design enables the reuse of partially correct proof structures while effectively combining high-level planning with fine-grained reasoning. We implement *HybridProver* in Isabelle/HOL and post-train two 7B-scale LLMs on our optimized Isabelle datasets. Experiments on the miniF2F Isabelle benchmark achieved a 73.8% success rate and improved upon the previous state of the art (61.9%), demonstrating that lightweight models, when combined with our approach, can effectively generate Isabelle/HOL proofs without relying on very large LLMs. Ablation studies further analyze the impact of dataset quality, training configurations, and sampling strategies on proof generation. All code and datasets are released.

1 Introduction

The field of formal methods plays a pivotal role in ensuring the reliability of critical systems across mathematics, computer science, and engineering (Bjørner and Havelund, 2014). By constructing machine-checkable proofs over formal specifications, interactive theorem provers (ITPs) provide strong guarantees of correctness. However, the process of developing formal proofs remains labor-intensive and requires substantial expertise, limiting the broader adoption of formal verification. To address this challenge, automated theorem proving has emerged as a paradigm to streamline formal verification (Barrett et al., 2011; De Moura and Bjørner, 2008; Groom, 2018; Schulz, 2002; Weidenbach et al., 1996), but its large proof-search space and limited adaptability to diverse proof scenarios have constrained its practicality.

Recent advances in large language models (LLMs) have opened new frontiers for automated theorem proving. Generating formal proofs can be viewed as a structured form of code generation, a setting in which LLMs have demonstrated strong capabilities (Chen et al., 2021; Jiang et al., 2024). Moreover, unlike natural language outputs, formal proofs can be mechanically verified by ITPs, e.g., Isabelle/HOL (Nipkow et al., 2002), Rocq (Sozeau et al., 2025), and Lean (Moura and Ullrich, 2021), providing a principled way to mitigate hallucination through external verification.

Existing LLM-based approaches to automated theorem proving typically fall into two categories: (1) whole-proof synthesis that produces complete formal proof attempts in a single pass (Xin et al., 2024; Liang et al., 2025; Wang et al., 2024c), and (2) tactic-based generation that constructs proofs via stepwise reasoning (Zhao et al., 2024; Jiang et al., 2023; Xu et al., 2026). Whole-proof synthesis excels at high-level planning but is error-prone at the level of detail, while tactic-based generation does the opposite. This suggests an opportunity to reuse the global structure of failed whole-proof attempts while repairing their local errors with tactic-level refinement.

In this work, we present *HybridProver*, an LLM-driven framework that integrates these two paradigms through an intermediate representation in the form of proof sketches. Given a theorem, *HybridProver* first generates candidate proofs using a whole-proof synthesis model. When these candidates are incorrect, *HybridProver* abstracts

them into proof sketches that retain the high-level structure while omitting low-level details. These sketches are then verified and refined using a tactic-based generation model, together with existing symbolic tools such as hammers (Blanchette et al., 2016), to produce a final whole proof.

This design highlights a general principle for LLM-based formal reasoning: separating high-level planning from low-level refinement and bridging them through verifiable proof sketches. Such sketches can still be checked for structural validity, enabling tactic-based generation to proceed under a well-formed proof context while reusing partially correct whole-proof attempts. As a result, *HybridProver* can flexibly compose different models and tools, combining neural generation with symbolic reasoning in a unified framework.

While the ideas for *HybridProver* can be implemented for many ITPs, we focus on Isabelle/HOL (Nipkow et al., 2002), using Sledgehammer (Paulson, 2012) as the hammer. Compared to prior LLM-based theorem-proving systems, which predominantly focus on Lean, Isabelle presents a complementary and relatively underexplored setting. It poses additional challenges for LLM-based proof generation due to its diverse proof styles and less standardized proof patterns, while also offering unique opportunities for constructing high-quality training data (see Section 2).

We evaluate *HybridProver* on the miniF2F benchmark (Zheng et al., 2022). Our approach improves proof success rates over prior Isabelle-based systems. We further show that combining whole-proof synthesis with tactic-based generation yields consistent performance gains, and that integrating symbolic tools such as Sledgehammer yields additional improvements. Beyond overall performance, our ablation studies highlight the importance of dataset quality, training strategies, and sampling diversity in LLM-based proof generation. To facilitate future research, we release our datasets and the code of *HybridProver*.[1]

Contributions:

- We propose *HybridProver*, a framework for LLM-based automated theorem proving that unifies whole-proof synthesis and tactic-based generation. By using proof sketches as intermediate representations, *HybridProver* connects high-level proof planning with low-level tactic refinement. It also enables imperfect whole-proof candidates to be reused and refined rather than discarded.
- We implement *HybridProver* in Isabelle/HOL and construct optimized training datasets. We further fine-tune two 7B-scale LLMs for both whole-proof synthesis and tactic-based generation, and enhance them with reinforcement learning guided by verification signals.
- *HybridProver* achieves a 73.8% success rate on miniF2F, improving over prior results for Isabelle-based systems(61.9%). We demonstrate that lightweight models can be effectively adapted to proof generation through high-level planning and low-level tactic refinement. Extensive ablations highlight the impact of data quality, training strategies, and sampling diversity.

2 Background: Why Isabelle?

We implement *HybridProver* in Isabelle/HOL, an interactive theorem prover (ITP) widely used for formal verification and mathematical reasoning (Nipkow et al., 2002). ITPs such as Isabelle, Rocq (Sozeau et al., 2025), and Lean (Moura and Ullrich, 2021) provide strong correctness guarantees by checking proofs with small, trusted kernels (Nipkow et al., 2002; Pierce et al., 2010). However, constructing formal proofs in these systems remains labor-intensive and requires significant expertise (Klein et al., 2009; Gu et al., 2016; Leroy et al., 2016). Our choice of Isabelle as a target ITP addresses its unique challenges while also taking advantage of its unique benefits.

Challenges of targeting Isabelle: broad use cases. Compared to prior LLM-based theorem proving systems, which predominantly focus on Lean, Isabelle remains relatively underexplored. This difference is partly due to the nature of existing benchmarks such as miniF2F, which are mathematical and better aligned with Lean's mathlib (Moura and Ullrich, 2021). In contrast, Isabelle is more widely used in system verification, where proofs vary significantly across domains and specifications. While Isabelle provides a large and diverse corpus of formal proofs (over 200K theorems, compared to Lean's 140K and Rocq's 71K (Drori et al., 2025; Reichel et al., 2023)). This diversity makes it challenging for LLMs to learn consistent proof patterns across domains.

[1] https://zenodo.org/records/20326385

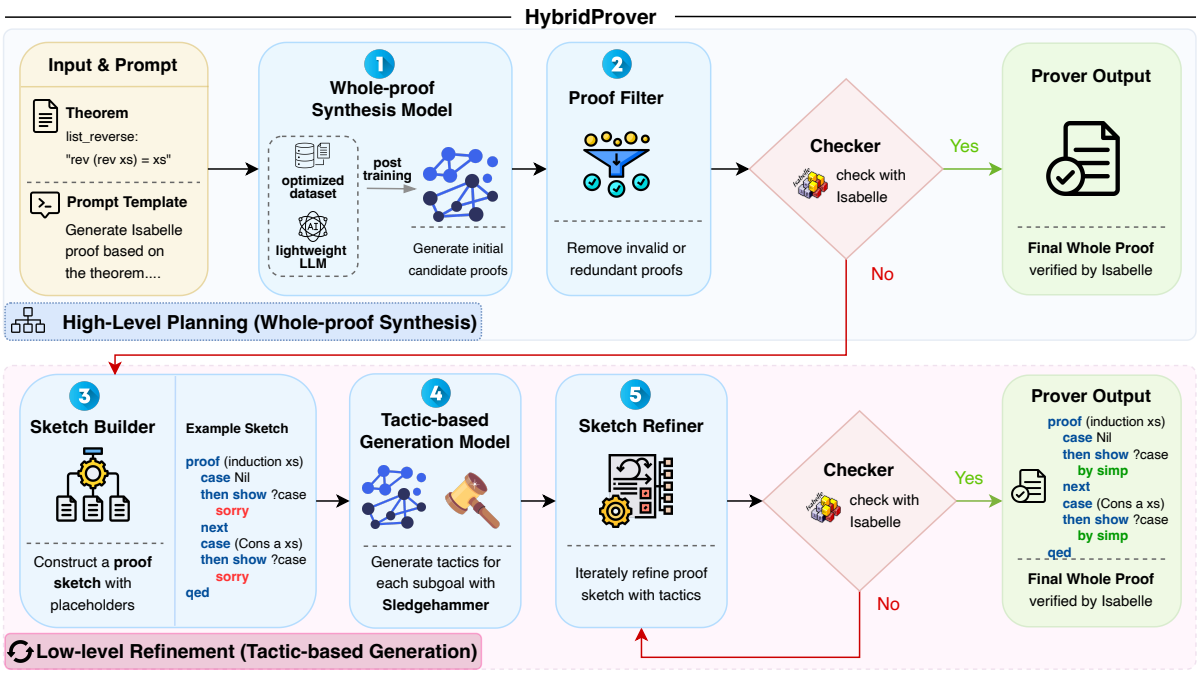

Figure 1: HybridProver Framework Overview

Advantages of targeting Isabelle: two proof styles. A distinctive feature of Isabelle is the coexistence of two proof styles: apply-style proofs (Wenzel, 2025) and Isar proofs (Wenzel and Team, 2025). Apply-style proofs construct derivations through stepwise tactic application, where a tactic is a primitive proof step that refines the current proof state to an updated state. While Isar proofs provide structured, human-readable proof scripts that resemble mathematical reasoning.

Listing 1: Isar and apply-style proof example

```
theorem list_reverse: "rev (rev xs) = xs"
Isar proof:                |apply-style proof:
proof(induction xs)        |apply(induction xs)
case Nil                   | apply auto
then show ?case by simp    | done
next case (Cons a xs)      |
then show ?case by simp    |
qed                        |
```

Listing 1 illustrates these two styles on the same theorem, which states that reversing a list twice does not change the list. Apply-style proofs are concise but opaque, requiring inspection of intermediate proof states, whereas Isar proofs explicitly expose the high-level structure.

Importantly, these two styles are complementary: Isar can serve as a high-level proof sketch, while apply-style tactics can discharge low-level subgoals. This observation motivates our design, where we treat Isar-style proofs as structured sketches and use tactic-based generation to refine them. By leveraging both styles, we construct a unified framework that bridges high-level planning and low-level reasoning, and enables more effective proof generation.

3 HybridProver

3.1 Framework Overview

Our framework is depicted in Figure 3. The workflow can be divided into the following steps:

Step 1: Prompt Construction. *HybridProver* takes as input a formal theorem statement in Isabelle syntax. The statement is concatenated with a structured prompt template (Listing 2) that specifies the task and output format, and is then fed into a post-trained *whole-proof synthesis model*. Trained on optimized Isabelle proof data, the model generates candidate proofs by capturing common proof patterns and basic reasoning structures.

Step 2: Whole-Proof Synthesis and Filtering. The whole-proof synthesis model generates n candidate proofs (we set $n = 128$ following prior work) in parallel to explore diverse proof paths. A lightweight *proof filter* is then applied to remove invalid or redundant outputs, reducing the cost of subsequent verification. The filter enforces basic syntactic constraints on Isabelle proofs. Outputs that do not conform to the required JSON format or do not start with valid Isabelle keywords (e.g.,

proof, *apply*, or *by*) are discarded. The remaining candidates are passed to the Isabelle checker for formal verification, noting that syntactic validity does not guarantee correctness.

With concurrent verification, multiple candidates can be checked simultaneously. If any candidate passes verification, the system returns the corresponding proof and terminates early. Otherwise, the valid candidates are forwarded to the next stage for further processing.

Step 3: Proof Sketch Construction. If none of the n candidate proofs passes the checker, the system invokes a *sketch builder* to transform syntactically valid candidates into proof sketches, rather than discarding them. The key idea is to lift detailed proof attempts to a higher-level representation that preserves global structure while abstracting away low-level tactic details.

Unlike prior work (Jiang et al., 2023), we construct proof sketches directly from LLM-generated candidate proofs. The sketch builder validates each subgoal proof independently: when checking one subgoal, it replaces the proofs of the remaining subgoals with *sorry*, allowing Isabelle to verify the current proof step under assumed completion of the others. Correct subgoal proofs are preserved, while failed ones are replaced with *sorry*. The resulting sketch is therefore a partially verified intermediate representation that reuses correct local proof steps and leaves only unresolved subgoals for refinement. The resulting sketches are then validated by the Isabelle checker to remove structurally invalid cases, such as incorrect proof methods or missing lemmas. Since low-level details are abstracted away, most sketches pass this stage. With concurrent verification, this process remains efficient.

Step 4: Subgoal Refinement and Tactic Generation. Given a proof sketch, we refine each subgoal marked with *sorry* into a verifiable proof step using a *tactic-based generation model*. This model combines a post-trained LLM for tactic generation with the Isabelle tool Sledgehammer.

For each unresolved subgoal, the LLM generates n candidate tactics in parallel, while Sledgehammer independently attempts to discharge the same subgoal. The resulting candidates are validated by the Isabelle checker. These two components are complementary, as each can solve subgoals that the other fails to prove. In contrast to whole-proof synthesis, which focuses on high-level structure and premise selection, this stage operates on localized subgoals and concrete proof steps. In our framework, premise selection is incorporated into the training of the whole-proof synthesis model, allowing the refinement stage to leverage richer contextual information for subgoal solving.

Algorithm 1 HybridProver workflow

Require: Theorem L, prompt template T
Ensure: Valid Isabelle proof or failure

$\mathcal{P} \leftarrow \mathrm{LLM}_{whole}(L, T)$ *// Step 1: whole-proof synthesis*
$\mathcal{P}_{syntax} \leftarrow proof_filter(\mathcal{P})$ *// Step 2: filtering*
$\mathcal{P}_{valid} \leftarrow \emptyset$
parallel for $p \in \mathcal{P}_{syntax}$ **do**
  if $\mathrm{check}(p, L)$ **then**
    $\mathcal{P}_{valid} \leftarrow \mathcal{P}_{valid} \cup \{p\}$
  end if end parallel
  if $\mathcal{P}_{valid} \neq \emptyset$ **then**
    return any $p \in \mathcal{P}_{valid}$
  end if
$\mathcal{S} \leftarrow \{\mathrm{replaceSorry}(p) \mid p \in \mathcal{P}\}$ *// Step 3: sketch construction*
$\mathcal{S}_{valid} \leftarrow \{s \in \mathcal{S} \mid \mathrm{check}(s, L)\}$
parallel for $s \in \mathcal{S}_{valid}$ **do** *// Steps 4–5: refinement*
  $G_1, \ldots, G_n \leftarrow \mathrm{parseSubgoals}(s)$
  $\mathcal{T} \leftarrow \bigcup_{i=1}^{n} \big(\mathrm{LLM}_{step}(G_i) \cup \mathrm{Sledgehammer}(G_i)\big)$
  $p_{refined} \leftarrow \mathrm{substituteSorry}(s, \mathcal{T})$
  if $\mathrm{IsabelleVerify}(p_{refined}, L)$ **then**
    return $p_{refined}$
  end if end parallel
return failure

Step 5: Proof Refinement. In the final stage, **sketch refiner** replaces the *sorry* placeholders in the proof sketch with the corresponding proof steps generated in Step 4, yielding a complete Isabelle proof. The refined proof is then submitted to the Isabelle checker for final verification. If verification fails, the system performs backtracking by substituting alternative tactics for unresolved subgoals and re-verifying the reconstructed proof. This process repeats until a valid proof is found or all candidates are exhausted. If successful, the system returns the verified proof; otherwise, it reports failure. This iterative refinement process bridges the gap between abstract proof sketches and complete proof code.

The overall workflow of *HybridProver* is summarized in Algorithm 1. Given a theorem L and a prompt template T, the whole-proof synthesis model generates candidate proofs, which are filtered and verified in parallel. If a valid proof is found, the system terminates early. Otherwise, syntactically valid candidates are converted into proof sketches and validated. For each valid sketch, subgoals are extracted and solved using a tactic-based generation model together with Sledgehammer. The resulting tactics are used to refine the sketch into a complete proof, which is then verified by Isabelle. This process iterates until a valid proof

is found or all candidates are exhausted. An Isabelle running example is shown in Appendix A.3.

3.2 Dataset optimization

We construct our Isabelle dataset from the Standard Library (STD) and the Archive of Formal Proofs (AFP) using PISA (Jiang et al., 2021). To reduce noise in raw formalization data, we apply a cleaning pipeline that extracts theorem–proof pairs, removes formatting artifacts (e.g., redundant whitespace), eliminates comments, and normalizes symbols. Each proof is then converted into a structured format, *{Isabelle_proof: proof code}*, to standardize the target output. We further partition the dataset according to Isabelle's two proof styles: Isar and apply-style proofs. This distinction aligns with our two-stage framework: Isar proofs capture high-level structure, while apply-style proofs focus on tactic-level reasoning.

To improve training efficiency and data quality, we filter examples based on proof length. Apply-style proofs in raw data are typically short (1–5 steps), so we retain only concise instances to encourage precise tactic generation. Isar proofs are longer (typically 11–30 steps), and we remove overly short or excessively long examples (outside 5–50 steps) to balance structural richness and learnability. The resulting dataset provides a cleaner and more structured training signal for both global proof generation and local tactic refinement.

3.3 Model Training

We first fine-tune two models corresponding to the two stages of *HybridProver*. The whole-proof synthesis model is trained on the Isar dataset to capture high-level proof structure, while the tactic-based generation model is trained on the apply-style dataset to generate fine-grained proof steps. Both models are obtained via supervised fine-tuning of pretrained models. To improve training efficiency and reduce memory consumption, we adopt LoRA (Hu et al., 2022), which freezes the base model and introduces low-rank trainable adapters. This allows us to scale to larger models while maintaining performance comparable to full fine-tuning under limited computational resources.

After supervised fine-tuning, we further optimize the model using reinforcement learning. We adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024), where Isabelle serves as the reward provider. The training data consists of a randomly sampled 1% subset of the optimized dataset. A key advantage of theorem proving is that generated proofs are formally verifiable, making Isabelle an accurate and reliable reward signal. Specifically, proofs that pass verification receive the highest reward, partially correct but invalid proofs receive a lower reward, and syntactically invalid or rejected outputs receive no reward. We additionally introduce penalties to discourage degenerate behaviors, such as generating placeholders (e.g., `sorry`) or trivial outputs. The final reward combines verification feedback with these structural penalties. This design encourages the model to generate complete, verifiable proofs while avoiding shortcuts that bypass actual reasoning.

Table 1: Dataset Sources and Configurations

Data Source	Purpose	Proof Style	Samples
AFP	Train (95%)	apply-style	116,308
		Isar-style	35,803
	Test (4%)	–	6,400
	Validation (1%)	–	1,600
STD	Train (100%)	apply-style	36,038
		Isar-style	9,018

4 Experiments

4.1 Setup

Dataset We construct the training dataset from the Isabelle Standard Library (STD) and the Archive of Formal Proofs (AFP) using PISA (Jiang et al., 2021), following the preprocessing described in Section 3.2. AFP is split into 95%/1%/4% for training/validation/test following prior work (Jiang et al., 2022, 2023), while STD is used entirely for training due to its dependency on AFP. The dataset statistics are summarized in Table 1. For evaluation, we use the miniF2F benchmark (Zheng et al., 2022), a widely used dataset consisting of 488 mathematical problems. Following prior work, we report results on the 244 test problems.

Experimental setup In initial small-model experiments, we generated proofs using more than 15 different open-source models. In the end, we chose Qwen-Coder-7B as our base model for further training because it had the highest proof success rate. For LLM fine-tuning and reinforcement learning(RL), we use 8 Nvidia A40 GPUs. Fine-tuning our models with LoRA on the whole optimized Isabelle dataset and 3 epochs takes around 200 GPU hours, while RL with GRPO takes around 100 GPU hours. The cost of ablation studies for other LLMs and configurations is not included. De-

tailed training configurations can be found in Appendix A.5. For evaluation, we use a 128-core Intel CPU with 256GB RAM while setting the concurrent processes to 64. The whole evaluation process cost nearly 10,000 CPU hours.

4.2 Results and Analysis

We evaluate *HybridProver* on the Isabelle miniF2F test set, using Sledgehammer as the baseline. Following prior work, we use a sampling size of 128 when generating candidate proofs. The results are shown in Table 2. Sledgehammer with heuristics achieves 20.9% success rate without LLM assistance. Recent LLM-based approaches significantly improve over this baseline, reaching 30%–50% success rates. With the integration of whole-proof synthesis and tactic-based generation, *HybridProver* achieves a success rate of 73.8%, outperforming prior Isabelle-based systems and improving over the previous best result of 61.9% from ProofAug (Liu et al., 2025) by 11.9%.

Table 2: Proof success rates on the Isabelle miniF2F

Method(using Isabelle)	Base Model	Test(%)
Sledgehammer	–	10.4
Sledgehammer + heuristics	–	20.9
Thor	–	29.9
Thor + expert iteration	–	35.2
DSP	Codex	39.3
POETRY	ProofGPT(1.3B)	42.2
Subgoal-Prover	GPT-3.5-Turbo	45.5
LEGO-Prover	GPT-3.5-Turbo	50.0
Lyra	GPT-4	51.2
SubgoalXL	Llama3-8B	56.1
ProofAug	Deepseek-math-7B	61.9
HybridProver	**Qwen-Coder-7B**	**73.8**

To analyze the contribution of each component, we evaluate several variants of *HybridProver* (Table 3). The whole-proof synthesis model alone achieves 45.9%, while the tactic-based generation model achieves 39.3%. A simple combination of the two models (taking the union of their results) improves the success rate to 49.6%. In miniF2F, some theorems are simple and require only a small number of proof steps. In such cases, the whole-proof synthesis model tends to generate verbose and often incorrect Isar proofs, while the tactic-based generation model is more effective at producing concise solutions. This explains the complementary behavior of the two models. Integrating Sledgehammer with heuristics further improves performance by approximately 2%, suggesting that the tactic-based LLM already covers most subgoals that can be solved by symbolic tools.

Table 3: Ablation results on Isabelle miniF2F

Variant	Test (%)
Whole-proof synthesis model	45.9
Tactic-based generation model	39.3
Whole-proof + tactic generation	49.6
Tactic generation + sketch refinement	71.3
HybridProver	**73.8**

Combining tactic-based generation with proof sketch refinement significantly boosts performance to 71.3%, demonstrating the effectiveness of reusing partially correct proofs. Interestingly, most sketches constructed from the whole-proof synthesis model are structurally correct: when evaluated without detailed tactics, sketches achieve up to 90% success rate. This indicates that high-level proof structure is often captured correctly even when local reasoning fails. Finally, *HybridProver* achieves a 73.8% success rate, showing that combining whole-proof synthesis with tactic-based refinement leads to substantial improvements, while symbolic tools provide additional, though smaller, gains. Besides, lightweight models can effectively generate Isabelle/HOL proofs when combined with our approach without relying on very large LLMs.

4.3 Additional Ablation Studies

RQ1: How do data quality and training parameters affect fine-tuning performance? We use the original Qwen-Coder-7B model as a baseline. Since it is only generically pretrained, it struggles to follow our task-specific instructions. To mitigate this, we prompt the model with a structured format (Listing 2) and enforce constrained outputs.

We then fine-tune the whole-proof model on variants of our optimized Isabelle dataset under different hyperparameter settings, including learning rate, LoRA rank, and number of training epochs. Our results show that data quality, learning rate, and training duration have significant impact on performance. In particular, training on raw data leads to noisy outputs with redundant tokens and formatting artifacts, which degrades proof generation quality. After applying the data cleaning procedure described in Section 3.2, the success rate on miniF2F improves by at least 10% across all training settings, highlighting the importance of dataset quality.

The ablation results for training parameters are shown in Figure 2. For training epochs, we find that 3 epochs perform best, while longer training leads to overfitting. This is partly due to the mismatch between the training data (covering mathematics, software, and systems) and the evaluation benchmark (miniF2F, which is purely mathematical). For the learning rate, we experiment with values ranging from 10^{-4} to 5×10^{-7} using a warm-up and cosine decay schedule, and find that 10^{-5} yields the best performance. Finally, we evaluate different LoRA ranks (8–256) and observe that the rank has minimal effect on performance, despite changing the number of trainable parameters. These findings provide practical guidance for fine-tuning LLMs in theorem proving tasks.

> **RA1:** Data quality and training hyperparameters (e.g., learning rate and epochs) significantly affect performance, while LoRA rank has limited impact.

RQ2: Does sampling diversity impact the results? We vary sampling parameters during inference, including temperature T and sampling rate. While proof generation requires stable outputs to ensure syntactic correctness, prior work adopts higher temperatures combined with multiple samples to improve success rates (e.g., T=1.0 in LISA and T=1.2 in Thor). In our setting, we evaluate the trained Qwen-Coder-7B model for whole proofs with $T \in [0.3, 1.2]$ and sampling rates from @32 to @128 (Figure 2). We observe that moderate temperatures (0.6–0.9) achieve the best performance, while both lower and higher values degrade success rates. Error analysis shows that excessively high T leads to frequent hallucination of non-existent facts, causing verification failures in Isabelle. We also find that increasing the sampling rate improves performance, indicating the benefit of broader exploration during inference.

> **RA2:** Sampling diversity influences results: moderate temperature works best, and higher sampling rates improve success rates.

RQ3: Does providing premises, few-shot examples, and labels help with proving? Many Isabelle theorems rely on auxiliary premises. We therefore include premises in the training data and train the model to both select relevant premises and construct proofs accordingly. This reduces *undefined fact* errors by around 50%, indicating that the model learns to avoid hallucinating nonexistent premises. However, this improvement does not translate into better performance on miniF2F, where most theorems do not require external premises.

For few-shot prompting, we add three representative examples (Listing 3) to the prompt, covering simple apply-style proofs, induction proofs, and Isar proofs. We observe no performance gain for either the trained Qwen-Coder-7B or the base model, suggesting that the model already captures the required syntax and proof patterns. We further augment each theorem with its source file path as a label, aiming to provide coarse-grained contextual information for premise selection. While this reduces the search space in principle (e.g., encouraging the model to retrieve premises from the corresponding theory file), it again yields negligible improvements on miniF2F, likely due to its homogeneous mathematical domain.

> **RA3:** Additional signals such as premises, few-shot examples, and labels provide limited gains on miniF2F.

5 Related Work

The use of LLMs to automate theorem proving was pioneered by Polu and Sutskever (Polu and Sutskever, 2020). Subsequent work has focused on improving the reasoning capabilities of LLM-based systems. Open-source interactive environments like Leandojo (Yang et al., 2023) for Lean, and PISA (Jiang et al., 2021) and Isabelle-client (Koepke et al., 2022) for Isabelle, make it easier to build LLM-based tools for automated theorem proving. Likewise, open-source datasets and benchmarks like miniF2F (Zheng et al., 2022) and Proofnet (Azerbayev et al., 2023) provide the basis for evaluation. Automation often uses premise selection prior to proof generation to find relevant premises (Yang et al., 2023; Jiang et al., 2022; Wang et al., 2024b; Mikuła et al., 2023). For example, LEGO (Wang et al., 2024b) builds a growing library and retrieves premises from it, while Thor (Jiang et al., 2022) delegates premise selection to hammers during the proving process. In *Hybrid-Prover*, we ask both the trained LLM and Sledgehammer in Isabelle to perform premise selection.

Prior work on LLM-based proof generation also

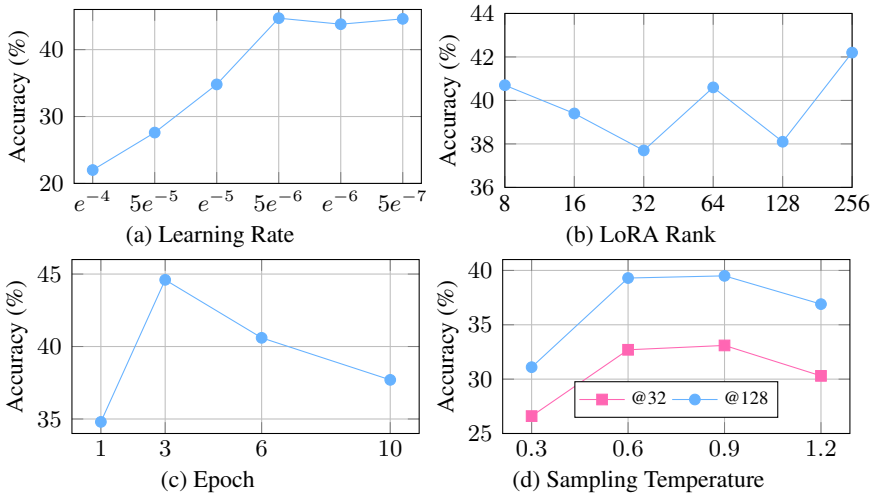

Figure 2: Ablation results under different training and sampling configurations.

varies by what the LLM is used for and by its input context. Some work (Yang et al., 2023; Zhao et al., 2024; Polu and Sutskever, 2020; Lin et al., 2024; Liang et al., 2025; Wang et al., 2024a; Achim et al., 2025) uses LLMs to generate tactic-based proofs step by step. Other work (Xin et al., 2024; First et al., 2023; Wang et al., 2024c; Ren et al., 2025; Shao et al., 2025; Wang et al., 2025; Lin et al., 2025; Chen et al., 2025; Xu et al., 2026) uses LLMs to generate the whole proof at once. For example, Polu (Polu and Sutskever, 2020) and FVEL (Lin et al., 2024) ask LLM to generate the next tactic based on the current proof state. Baldur (First et al., 2023) makes LLMs generate whole proofs based on the input theorem. *HybridProver* organically combines these two approaches and gets a better proof quality. After the proof generation, some works also focus on proof search that chooses appropriate generated responses with different algorithms, such as Best first search (Polu and Sutskever, 2020; Han et al., 2022; Yang et al., 2023; Xin et al., 2025), and Monte Carlo tree search (Wang et al., 2023; Lample et al., 2022; Xin et al., 2024). Despite the improvement, these search algorithms often require additional computational resources that are non-trivial. Besides, a few works try to integrate hammers and sketches to improve the performance (Jiang et al., 2022, 2023; Mikuła et al., 2023; Zhao et al., 2024; Yang et al., 2023; Liu et al., 2025). For instance, DSP (Jiang et al., 2023) generates proof sketches with subgoals and uses hammers to boost the proof success rate. Recent work (Ospanov et al., 2025; Ding et al., 2026) has also explored agent-based approaches to enhance the reasoning capabilities of LLMs.

HybridProver differs from them in three aspects. First, Most prior work focuses on Lean, while we study another important ITP called Isabelle. Second, we not only integrate hammers but also post-train LLMs based on our optimized dataset to improve the performance. Third, we build the proof sketch mechanically from the whole proof, which bridges the gap between whole-proof synthesis and tactic-based generation, while other work, such as DSP, builds sketches from informal proofs by prompting another LLM.

6 Conclusion and Future Work

We introduced *HybridProver*, an LLM-driven framework for automated theorem proving that unifies whole-proof synthesis and tactic-based generation through proof sketches as an intermediate representation. This design enables effective interaction between global planning and local reasoning, further enhanced by automated tools such as hammers. In addition, we construct optimized Isabelle datasets and train LLMs via both supervised fine-tuning and reinforcement learning guided by formal verification signals. Our implementation of *HybridProver* in Isabelle achieves a new state of the art on the miniF2F benchmark, reaching 73.8% accuracy and significantly outperforming the previous best result of 61.9%.

In future work, we plan to integrate more advanced proof search techniques into the tactic-based generation process to improve efficiency. We also plan to develop a more user-friendly integrated system to facilitate practical adoption.

Limitations

One limitation of *HybridProver* lies in its relatively simple proof search strategy during the proof sketch refinement stage. As the number of subgoals increases, the refinement and evaluation process becomes increasingly inefficient, leading to substantial computational overhead. Moreover, *HybridProver* relies heavily on the underlying theorem prover (e.g., Isabelle) as a verification oracle. While this provides accurate and reliable feedback, it also introduces a bottleneck in terms of latency and scalability, especially when evaluating a large number of candidate proofs. Another limitation is that the model is trained on existing proof corpora, which may bias it toward familiar proof patterns. As a result, its ability to generalize to novel or significantly different proof structures remains limited.

References

Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, Martin Michelsen, Riley Patterson, Eric Rodriguez, Laura Scharff, Vikram Shanker, Vladmir Sicca, Hari Sowrirajan, Aidan Swope, Matyas Tamas, and 4 others. 2025. Aristotle: Imo-level automated theorem proving. *Preprint*, arXiv:2510.01346.

Sidney Amani, Alex Hixon, Zilin Chen, Christine Rizkallah, Peter Chubb, Liam O'Connor, Joel Beeren, Yutaka Nagashima, Japheth Lim, Thomas Sewell, and 1 others. 2016. Cogent: Verifying high-assurance file system implementations. *ACM SIGARCH Computer Architecture News*, 44(2):175–188.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and Jeremy Avigad. 2023. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*.

Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. cvc4. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 171–177. Springer.

Dines Bjørner and Klaus Havelund. 2014. 40 years of formal methods. In *FM 2014: Formal Methods*, pages 42–61, Cham. Springer International Publishing.

Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C Paulson, and Josef Urban. 2016. Hammering towards qed. *Journal of Formalized Reasoning*, 9(1):101–148.

Tej Chajed, Joseph Tassarotti, M Frans Kaashoek, and Nickolai Zeldovich. 2019. Verifying concurrent, crash-safe systems with perennial. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 243–258.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, Cheng Ren, Jiawei Shen, Wenlei Shi, Tong Sun, He Sun, Jiahui Wang, Siran Wang, Zhihong Wang, Chenrui Wei, and 17 others. 2025. Seed-prover: Deep and broad reasoning for automated theorem proving. *Preprint*, arXiv:2507.23726.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. 2023. Nl2tl: Transforming natural languages to temporal logics using large language models. *arXiv preprint arXiv:2305.07766*.

Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *International Conference on Computer Aided Verification*, pages 383–396. Springer.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Haoran Ding, Zhaoguo Wang, and Haibo Chen. 2026. Fm-agent: Scaling formal methods to large systems via llm-based hoare-style reasoning. *Preprint*, arXiv:2604.11556.

Iddo Drori, Gaston Longhitano, Mao Mao, Seunghwan Hyun, Yuke Zhang, Sungjun Park, Zachary Meeks, Xin-Yu Zhang, Ben Segev, Howard Yong, Nakul Verma, Avi Shporer, Alon Amit, and Madeleine Udell. 2025. Diverse inference and verification for advanced reasoning. *Preprint*, arXiv:2502.09955.

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

Nick Groom. 2018. *The Vampire*. Yale University Press.

Ronghui Gu, Jérémie Koenig, Tahina Ramananandro, Zhong Shao, Xiongnan Wu, Shu-Chun Weng, Haozhong Zhang, and Yu Guo. 2015. Deep specifications and certified abstraction layers. *ACM SIGPLAN Notices*, 50(1):595–608.

Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan Newman Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. 2016. {CertiKOS}: An extensible architecture for building certified concurrent {OS} kernels. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 653–669.

Ronghui Gu, Zhong Shao, Jieung Kim, Xiongnan Wu, Jérémie Koenig, Vilhelm Sjöberg, Hao Chen, David Costanzo, and Tahina Ramananandro. 2018. Certified concurrent abstraction layers. *ACM SIGPLAN Notices*, 53(4):646–661.

Christopher Hahn, Frederik Schmitt, Julia J Tillman, Niklas Metzger, Julian Siber, and Bernd Finkbeiner. 2022. Formal specifications from natural language. *arXiv preprint arXiv:2206.01962*.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W Ayers, and Stanislas Polu. 2022. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Jilin Hu, Fanlang Zeng, Yongwang Zhao, Zhuoruo Zhang, Leping Zhang, Jianhong Zhao, Rui Chang, and Kui Ren. 2024. ProveriT: A Parameterized, Composable, and Verified Model of TEE Protection Profile . *IEEE Transactions on Dependable and Secure Computing*, 21(06):5341–5358.

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

Albert Qiaochu Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu, and Mateja Jamnik. 2022. Thor: Wielding hammers to integrate language models and automated theorem provers. *Advances in Neural Information Processing Systems*, 35:8360–8373.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. 2023. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *Preprint*, arXiv:2406.00515.

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, and 1 others. 2009. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220.

Peter Koepke, Anton Lorenzen, and Boris Shminke. 2022. Cicm'22 system entries. In *Intelligent Computer Mathematics*, pages 344–348, Cham. Springer International Publishing.

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

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

Zhenwen Liang, Linfeng Song, Yang Li, TAO YANG, Haitao Mi, and Dong Yu. 2025. Mps-prover: Advancing stepwise theorem proving by multi-perspective search and data curation. In *Advances in Neural Information Processing Systems*, volume 38, pages 137891–137916. Curran Associates, Inc.

Xiaohan Lin, Qingxing Cao, Yinya Huang, Haiming Wang, Jianqiao Lu, Zhengying Liu, Linqi Song, and Xiaodan Liang. 2024. Fvel: Interactive formal verification environment with large language models via theorem proving. *Advances in Neural Information Processing Systems*, 37:54932–54946.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. 2025. Goedel-prover: A frontier model for open-source automated theorem proving. *Preprint*, arXiv:2502.07640.

Haoxiong Liu, Jiacheng Sun, Zhenguo Li, and Andrew C Yao. 2025. Proofaug: efficient neural theorem proving via fine-grained proof structure analysis. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org.

Maciej Mikuła, Szymon Tworkowski, Szymon Antoniak, Bartosz Piotrowski, Albert Q Jiang, Jin Peng Zhou, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. 2023. Magnushammer: A transformer-based approach to premise selection. In *The Twelfth International Conference on Learning Representations*.

Leonardo de Moura and Sebastian Ullrich. 2021. 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*, pages 625–635. Springer.

Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer.

Azim Ospanov, Farzan Farnia, and Roozbeh Mohit. 2025. Apollo: Automated llm and lean collaboration for advanced formal reasoning. In *Advances in Neural Information Processing Systems*, volume 38, pages 41599–41633. Curran Associates, Inc.

Lawrence Paulson. 2012. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In *PAAR-2010: Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning*, volume 9 of *EPiC Series in Computing*, pages 1–10. EasyChair.

Benjamin C Pierce, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, and Brent Yorgey. 2010. Software foundations. *Webpage: http://www. cis. upenn. edu/bcpierce/sf/current/index. html*, 16.

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

Tom Reichel, R Henderson, Andrew Touchet, Andrew Gardner, and Talia Ringer. 2023. Proof repair infrastructure for supervised models: Building a large proof repair dataset. In *14th International Conference on Interactive Theorem Proving (ITP 2023)*, pages 26–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. 2025. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *Preprint*, arXiv:2504.21801.

Stephan Schulz. 2002. E–a brainiac theorem prover. *Ai Communications*, 15(2-3):111–126.

Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. 2025. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *Preprint*, arXiv:2511.22570.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Shweta Shinde, Shengyi Wang, Pinghai Yuan, Aquinas Hobor, Abhik Roychoudhury, and Prateek Saxena. 2020. {BesFS}: A {POSIX} filesystem for enclaves with a mechanized safety proof. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 523–540.

Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. 2018. Learning loop invariants for program verification. *Advances in Neural Information Processing Systems*, 31.

Xujie Si, Aaditya Naik, Hanjun Dai, Mayur Naik, and Le Song. 2020. Code2inv: A deep learning framework for program verification. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part II 32*, pages 151–164. Springer.

Helgi Sigurbjarnarson, James Bornholt, Emina Torlak, and Xi Wang. 2016. {Push-Button} verification of file systems via crash refinement. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 1–16.

Matthieu Sozeau, Yannick Forster, Meven Lennon-Bertrand, Jakob Nielsen, Nicolas Tabareau, and Théo Winterhalter. 2025. Correct and complete type checking and certified erasure for coq, in coq. *Journal of the ACM*, 72(1):1–74.

Freek Verbeek, Oto Havle, Julien Schmaltz, Sergey Tverdyshev, Holger Blasum, Bruno Langenstein, Werner Stephan, Burkhart Wolff, and Yakoub Nemouchi. 2015. Formal api specification of the pikeos separation kernel. In *NASA Formal Methods: 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings 7*, pages 375–389. Springer.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. TRL: Transformers Reinforcement Learning.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, and 21 others. 2025. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *Preprint*, arXiv:2504.11354.

Haiming Wang, Huajian Xin, Zhengying Liu, Wenda Li, Yinya Huang, Jianqiao Lu, Zhicheng Yang, Jing Tang, Jian Yin, Zhenguo Li, and Xiaodan Liang. 2024a. Proving theorems recursively. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, and 1 others. 2024b. Lego-prover: Neural theorem proving with growing libraries. In *12th International Conference on Learning Representations (ICLR 2024)*. International Conference on Learning Representations, ICLR.

Haiming Wang, Ye Yuan, Zhengying Liu, Jianhao Shen, Yichun Yin, Jing Xiong, Enze Xie, Han Shi, Yujun Li, Lin Li, and 1 others. 2023. Dt-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12632–12646.

Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. 2018. First experiments with neural translation of informal to formal mathematics. In *Intelligent Computer Mathematics: 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings 11*, pages 255–270. Springer.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. 2024c. Theoreml-lama: Transforming general-purpose llms into lean4 experts. *arXiv preprint arXiv:2407.03203*.

Christoph Weidenbach, Bernd Gaede, and Georg Rock. 1996. Spass & flotter version 0.42. In *Automated Deduction—Cade-13: 13th International Conference on Automated Deduction New Brunswick, NJ, USA, July 30–August 3, 1996 Proceedings 13*, pages 141–145. Springer.

Cheng Wen, Jialun Cao, Jie Su, Zhiwu Xu, Shengchao Qin, Mengda He, Haokun Li, Shing-Chi Cheung, and Cong Tian. 2024. Enchanting program specification synthesis by large language models using static analysis and program verification. In *International Conference on Computer Aided Verification*, pages 302–328. Springer.

Makarius Wenzel. 2021. The isabelle system manual.

Makarius Wenzel. 2025. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic (Tutorial)*. Technische Universität München.

Makarius Wenzel and Isabelle Development Team. 2025. *The Isabelle/Isar Reference Manual*. Technische Universität München.

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

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, and 1 others. 2024. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *CoRR*.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Ming Ding. 2025. BFS-prover: Scalable best-first tree search for LLM-based automatic theorem proving. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32588–32599, Vienna, Austria. Association for Computational Linguistics.

Qiyuan Xu, Renxi Wang, Peixin Wang, Haonan Li, and Conrad Watt. 2026. A minimalist proof language for neural theorem proving over isabelle/hol. *Proc. ACM Program. Lang.*, 10(OOPSLA1).

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

Xueliang Zhao, Lin Zheng, Haige Bo, Changran Hu, Urmish Thakker, and Lingpeng Kong. 2024. Subgoalxl: Subgoal-based expert learning for theorem proving. *arXiv preprint arXiv:2408.11172*.

Yongwang Zhao, David Sanán, Fuyuan Zhang, and Yang Liu. 2017. Refinement-based specification and security analysis of separation kernels. *IEEE Transactions on Dependable and Secure Computing*, 16(1):127–141.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2022. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *ICLR*.

A Technical Appendices and Supplementary Material

A.1 Hammers

We integrate hammers in our framework to augment theorem proving. Blanchette et al. (Blanchette et al., 2016) define hammers as methods that "automate reasoning over large libraries developed with formal proof assistants." Most ITPs have their own hammers, e.g., Coqhammer in Rocq and Sledgehammer in Isabelle. Sledgehammer in Isabelle is the most powerful one (Paulson, 2012). Providing the input theorem, Sledgehammer first finds relevant premises using machine learning and a traditional filtering algorithm (Paulson, 2012). Then it translates the input theorem together with a heuristic selection of hundreds of premises into goals that external ATPs can recognize. Then, ATPs solve these problems and return the results to Sledgehammer. Sledgehammer thus gets the useful premises and finally reconstructs proofs for the Isabelle trusted kernel to check. Every ITP has a trusted kernel, a minimal and rigorously verified core that checks manually constructed proofs following mathematical logic rules (Nipkow et al., 2002; Pierce et al., 2010). Since the kernel checks these proofs, it is unnecessary for the user to worry about the correctness of the generated proof.

Sledgehammer makes writing proofs convenient. Isabelle users only need to write the assumptions and the final goal that they want to prove. Then

users can call Sledgehammer directly to finish the proof. Sledgehammer is good at generating short proofs (Paulson, 2012). Thus, if users also provide a proof sketch with some subgoals introducing how the proof might be done, Sledgehammer will find proofs for each subgoal and construct the whole proof more easily. It is worth mentioning that Sledgehammer does not do induction very well. LLMs might help with theorems that require induction proofs.

A.2 Explanation of Isabelle Code

Listing 1 shows two Isabelle proofs of the same theorem, which states that reversing a list twice does not change the list. In the apply-style proof (right), we first apply induction on the list *xs*, then we use a tactic *auto* in Isabelle to finish the proof. The keyword *done* marks the end of the proof. This proof style is concise, but to understand what these tactics are doing, we need to check the proof state of each step in Isabelle. In the Isar proof (left), we first apply induction on list *xs* that clearly splits the goals into two cases, i.e., case *Nil* and case *Cons a xs*. Here *Nil* means the empty list, and *Cons* means appending an element *a* to the front of the original list *xs*.

Note that these two styles of proofs can be combined, with Isar sketching the final proof while apply-style tactics discharge the remaining subgoals to fill in the sketch. For example, for the proof in Listing 1, a user could write a proof sketch in Isar by writing *sorry* instead of *by simp*—telling Isabelle to assume the subgoal is correct. Then, after using Isabelle to check that the proof sketch is sufficient, users can fill in the details to finish the proof.

A.3 A Running Example of HybridProver

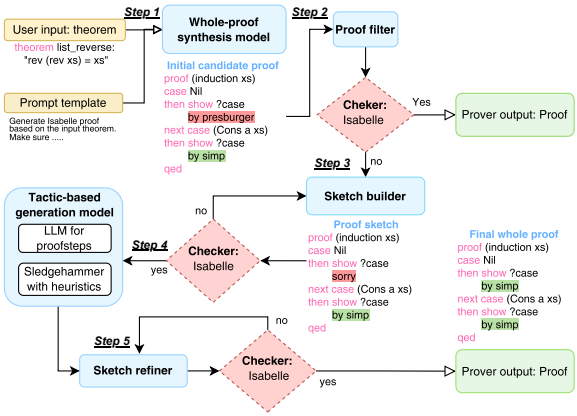

Figure 3: HybridProver running example demonstration

As a running example in Figure 3, we use the Isabelle theorem *list_reverse*, which states that reversing a list twice yields the original list: `rev (rev xs) = xs`. The initial candidate proof generated by the whole-proof synthesis model follows an Isar-style induction structure. It starts with `proof (induction xs)`, which splits the theorem into the base case `Nil` and the inductive case `Cons a xs`. In the candidate proof, the base case is discharged by `by presburger`, while the inductive case is discharged by `by simp`.

However, this candidate proof does not pass the Isabelle checker, because `by presburger` is not a suitable tactic for proving the `Nil` case of this list theorem. Although the overall proof structure is reasonable—it correctly uses induction and introduces the two standard list cases—one of its local proof steps is incorrect. This example illustrates why directly discarding the whole candidate proof would waste useful structure.

HybridProver therefore constructs a proof sketch from this failed candidate. In the sketch shown in Figure 3, the concrete tactics `by presburger` are replaced with `sorry`, while the correct subgoal proof and surrounding Isar structure is preserved: `proof (induction xs)`, `case Nil`, `next case (Cons a xs)`, and `qed` remain unchanged. The resulting sketch keeps the high-level induction argument but leaves the local proof obligations unresolved.

During refinement, the tactic-based generation model and Sledgehammer search for concrete tactics to replace the `sorry` placeholders. For this example, the first subgoal can be solved by `by simp`. After substitution, the final proof replaces each `sorry` with `by simp`, producing a complete Isabelle proof that passes verification: the base case `Nil` is proved by simplification, and the inductive case `Cons a xs` is also discharged by simplification.

This example shows that the initial whole-proof candidate already contains a useful high-level structure, even though one local tactic is wrong. By preserving the Isar structure and refining only the unresolved proof steps, HybridProver converts the failed candidate into a verified proof.

A.4 Additional Related Work

Interactive theorem proving. Theorem proving is widely used for ensuring the safety and security of critical software and systems. Most work in interactive theorem proving involves some manual

proof construction by human experts, a process that is labor-intensive and time-consuming. For example, the seL4 team (Klein et al., 2009) takes 30 person-years to verify a general-purpose microkernel with 8700 lines of C code in Isabelle. The CertiKOS team (Gu et al., 2016) verifies a concurrent OS through deep abstraction layers in Rocq (Gu et al., 2015, 2018). They take 3 person-years to prove 6500 lines of C and x86 assembly. CompCert (Leroy et al., 2016) is the first formally verified C compiler to be exempt from miscompilation. Six person-years are taken in Rocq to construct proofs. Additionally, most theorem-proving works in other operating systems take more than 1 person-year (Sigurbjarnarson et al., 2016; Shinde et al., 2020; Chajed et al., 2019; Zhao et al., 2017; Amani et al., 2016; Hu et al., 2024; Verbeek et al., 2015). Automated theorem provers (ATPs) such as Z3 and CVC4 offer full automation (Barrett et al., 2011; De Moura and Bjørner, 2008; Groom, 2018; Schulz, 2002; Weidenbach et al., 1996), but are typically limited to first-order logic and suffer from search-space explosion. This motivates the use of LLMs to improve the usability of ITPs without sacrificing their expressiveness and scalability.

Autoformalization. In this work, we focus on theorem proving where a formal specification already exists. Another interesting area is autoformalization, where researchers translate natural language texts into formal specifications and proofs (Azerbayev et al., 2023; Wang et al., 2018; Cosler et al., 2023; Hahn et al., 2022; Wu et al., 2022; Chen et al., 2023). The amount of natural language texts and mathematical proofs is far greater than formal texts. Therefore, autoformalization creates a lot of high-quality data used for model training, which solves the data scarcity problem. Other works also try to generate specifications based on code (Wen et al., 2024; Cosler et al., 2023; Si et al., 2018, 2020).

Premise Selection. Premises refer to the auxiliary lemmas, as well as datatype and function definitions, that are referenced to establish the target theorem. Existing studies on premise selection mainly follow two technical routes. One line of work builds an external premise retrieval library and retrieves relevant premises according to semantic similarity or symbolic matching with the input theorem (Wang et al., 2024b; Yang et al., 2023). The other incorporates premise usage into model training, enabling the model to memorize common premises and their contexts and to invoke them automatically when proving similar theorems (First et al., 2023). We adopt the latter approach by integrating premise selection into the training objective of the whole-proof synthesis model. As a result, the model is encouraged to identify and insert necessary premises during proof generation, thereby providing richer contextual support for the subsequent strategy generation stage and improving the accuracy of subgoal solving.

A.5 Additional Setup

Dataset. PISA includes the STD and AFP with version 2021-10-22. The former is more foundational, with around 70K theorems. The latter is a proof library contributed by most Isabelle researchers, with around 170K theorems. It diverges from mathematical problems to formal verification of software and hardware.

MiniF2F (Zheng et al., 2022) is used in many research studies (Yang et al., 2023; Jiang et al., 2023; Zhao et al., 2024). It is an open-source collection containing 488 high school math competition problems. These problems are divided into two groups: 244 for validation and 244 for final testing. All problems have been translated into 4 proof assistant languages: Lean, HOL light, Metamath, and Isabelle.

Isabelle Interactive environment. We interact with Isabelle using Isabelle-client (Koepke et al., 2022), which provides detailed developer documentation and Isabelle interface descriptions, making it easy to use and maintain. It is a Python client library that strictly follows the Isabelle system manual (Wenzel, 2021). To improve system throughput and efficiency, we employ a concurrent architecture based on a single server, a single client, and multiple sessions. Our experiments show that using multiple Isabelle server processes leads to resource contention. By contrast, multiple independent sessions can load different Isabelle theory libraries separately, thereby supporting the verification of different theorems in the evaluation dataset. In addition, we found that a multi-client single-session setting is error-prone and incurs high resource consumption, and therefore it is not adopted in our system.

watchdog_timeout is set to 30s for evaluation. Evaluation in Isabelle is time-consuming even for a small sampling rate and a small dataset. It costs 300 CPU hours to evaluate 244 theorems in miniF2F

with a 32 sampling rate for one LLM. Thus, we do ablation studies with a relatively low sampling rate and small timeout. To set up Sledgehammer, we follow the default Isabelle2021 configuration (Jiang et al., 2021), where the Sledgehammer timeout limit is 30s, with the on-machine five default ATPs (E, SPASS, Vampire, Z3, and CVC4). We also complement Sledgehammer with 11 heuristics for proofs: *['by auto', 'by simp', 'by blast', 'by fastforce', 'by force', 'by eval', 'by presburger', 'by sos', 'by arith', 'by linarith', 'by (auto simp: field_simps)']*.

Training Configurations For fine-tuning, we use LoRA (Hu et al., 2022) to train around 1% of all parameters with a *lora_rank* 32 and all target layers open for adjustment. In the first 10% of steps, the learning rate warms up linearly from 0 to the maximum value e^{-5}. Then it decays to 0 following a cosine schedule. The context length for the training data is 2048 tokens, and the total batch size is 64.

We implement reinforcement learning using the GRPO trainer from the TRL framework (von Werra et al., 2020). For each input theorem, we sample 16 candidate proofs in parallel. The reward function combines two components: (1) a verification reward obtained from Isabelle, and (2) a penalty for undesirable patterns. Concretely, proofs that pass Isabelle checking receive a reward of 1.0, proofs that are syntactically valid but fail verification receive 0.2, and invalid outputs receive 0.0. Additionally, outputs containing tokens such as `sorry` or malformed proof patterns are assigned a negative penalty (-0.2), discouraging trivial or incomplete proofs. We train the model for 3 epochs with a learning rate of 2×10^{-5}, using a per-device batch size of 2.

A.6 Prompt template and few-shot examples

Listing 2: Prompt template

```
Generate proof code in Isabelle/HOL
based on the input Isabelle/HOL
theorem. Make sure that the
generated proof can be verified by
Isabelle/HOL. Do not generate extra
natural language descriptions.
**Input theorem:**
```

Listing 3: Few-shot examples

```
Input theorem 1:
theorem app_Nil: "[] @ xs = (xs :: '
a list)"
Isabelle proof 1:
{Isabelle proof: by simp}

Input theorem 2:
theorem list_reverse: "rev (rev xs)
= xs"
Isabelle proof 2:
{Isabelle proof: apply (induct xs)
apply simp apply simp done}

Input theorem 3:
theorem append_assoc: "(xs @ ys) @
zs = xs @ (ys @ zs)"
Isabelle proof 3:
{Isabelle proof: proof (induct xs)
case Nil then show ?case by simp
next case (Cons x xs) then show ?
case by simp qed}
```

A.7 Additional ablation results

RQ4: Is the proof pattern similarity equal to the proof quality? During the fine-tuning stage, we try to feed the model enough theorem-proof pairs. The LLM learns to generate proof patterns similar to the training data. But is the proof pattern similarity equal to the quality? Thus, we not only test the success rate of the miniF2F dataset, but also record all responses from the sampling during the whole evaluation. We make a comparison between them and find that generally, the higher the total correct response is, the higher the success rate for all theorems is. It demonstrates that proof pattern similarity is strongly relevant to the proof quality, reflecting the improvement from training.

RA4: The more similar an LLM-generated proof pattern is to the training data, the higher the quality of the proof.