PROOFCOMPASS: Enhancing Specialized Provers with LLM Guidance

Nicolas Wischermann [1] **Claudio Mayrink Verdun** [2] **Gabriel Poesia** [3] **Francesco Noseda** [1]

Abstract

Language models have become increasingly powerful tools for formal mathematical reasoning. However, most existing approaches rely exclusively on either large general-purpose models or smaller specialized models, each with distinct limitations, while training specialized large models still requires significant computational resources. This paper introduces PROOFCOMPASS, a novel hybrid methodology that achieves remarkable computational efficiency by strategically guiding existing specialized prover methods, such as DeepSeek-Prover-v1.5-RL (DSP-v1.5) with a Large Language Model (LLM) without requiring additional model training. The LLM provides natural language proof strategies and analyzes failed attempts to select intermediate lemmas, enabling effective problem decomposition. On the miniF2F benchmark, PROOFCOMPASS demonstrates substantial resource efficiency: it outperforms DSP-v1.5 ($54.9\% \rightarrow 55.3\%$) while using 25x fewer attempts ($3200 \rightarrow 128$). Our synergistic approach paves the way for simultaneously improving computational efficiency and accuracy in formal theorem proving.

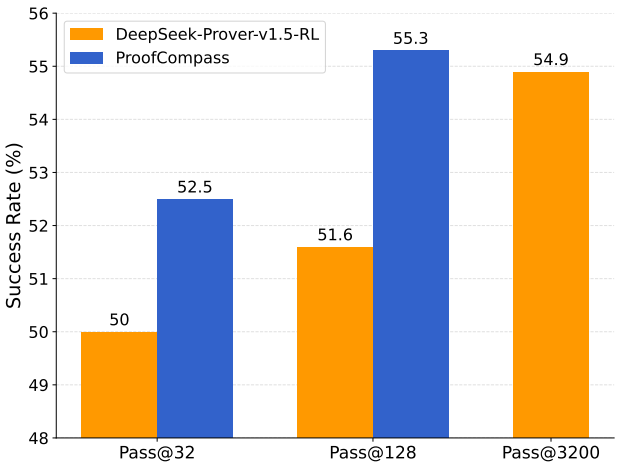

Figure 1. Comparison of PROOFCOMPASS versus DeepSeek-Prover-v1.5-RL (Xin et al., 2024) on the miniF2F-test dataset. Each group of bars shows the side-by-side success rates for a specific Pass@k budget. The figure highlights the resource efficiency of PROOFCOMPASS, showing its Pass@128 performance is comparable to the baseline's more computationally expensive Pass@3200 result.

1. Introduction

The formalization of mathematics, where one seeks to translate mathematical proofs into machine-verifiable form, represents a crucial step toward more reliable mathematical knowledge (Avigad, 2024). While proof assistants like Isabelle (Paulson, 1994), HOL Light (Harrison, 2009), and Lean (de Moura & Ullrich, 2021) have made formalization possible, the process has historically been labor intensive, requiring deep expertise in both mathematics and formal verification tools. Recent advances in large language models (LLMs) show promise in dramatically changing this landscape, with formal mathematical reasoning emerging as a crucial new frontier (Yang et al., 2024) and prompting broader reflection on the future of mathematical practice (Venkatesh, 2024; Fraser et al., 2024). For instance, models like GPT-4 (Achiam et al., 2023) have shown remarkable mathematical reasoning capabilities on complex problems. At the same time, specialized models have achieved unprecedented success in formal mathematics, with the recently released (April 30) DeepSeek-Prover-V2 (Ren et al., 2025), for instance, automatically solving 88.9% of the problems on miniF2F (Zheng et al., 2021), a central benchmark in formal mathematics.

Recent efforts to leverage language models for automated theorem proving have largely followed two distinct paths, each with its own set of strengths and limitations. One path focuses on using large, general-purpose language models, often by employing them within structured, multi-stage algorithms (Jiang et al., 2023) that break down the task of formal proof generation. While these large models have shown strong informal mathematical reasoning capabilities,

[1]Federal University of Rio de Janeiro [2]Harvard John A. Paulson School of Engineering and Applied Sciences [3]Stanford University. Correspondence to: Nicolas Wischermann <ndasilva@ufrj.br>.

The second AI for MATH Workshop at the 42nd International Conference on Machine Learning, Vancouver, Canada.

they frequently struggle with the precise syntax and tactics required by proof assistants. Alternatively, a significant body of work has focused on developing specialized models, often at a smaller scale (e.g., 7B parameters or less), tailored specifically for formal mathematics (Welleck & Saha, 2023; Xin et al., 2024; Lin et al., 2025; Xin et al., 2025). These models excel at generating syntactically correct proofs and closing smaller subgoals, but their relatively modest size inherently limits their mathematical reasoning ability.

More recently, approaches like Kimina-Prover (72B) (Wang et al., 2025) and DeepSeek-Prover-V2 (671B) (Ren et al., 2025) have achieved state-of-the-art results by training large models specifically for theorem proving. However, this paradigm shift towards ever-larger specialized models presents significant challenges. The prohibitive computational resources required for training are largely inaccessible to the broader research community, creating a high barrier to entry and making it difficult to contribute to the advancement of the frontier in AI-assisted theorem proving and formal mathematics.

In this work, we introduce PROOFCOMPASS: a synergistic methodology that leverages the broad reasoning capabilities of a large language model to guide a specialized prover, without requiring model training. This approach proves to be highly efficient, matching the average performance of a standalone specialized prover while using a 25-fold smaller computational budget. Furthermore, the framework is designed for modularity: the guiding LLM can be readily swapped, and its core principles are adaptable for use with other specialized provers. By substantially reducing the resources needed to achieve superior results, our work directly addresses the high barrier to entry created by resource-intensive models. This offers a more accessible path forward, helping to democratize cutting-edge research in automated theorem proving.

Concretely, we use a broadly capable LLM to direct the specialized DeepSeek-Prover-v1.5-RL (DSP-v1.5) model (Xin et al., 2024), augmenting its theorem-proving ability in Lean 4 (de Moura & Ullrich, 2021) through two primary mechanisms. First, we leverage the LLM's superior natural language understanding and mathematical reasoning abilities to produce a detailed natural language (NL) proof, which, after summarization, is provided to DSP-v1.5 to guide it during formal proof generation. Second, inspired by approaches that utilize intermediate goals or lemmas to structure proof search (Wang et al., 2024b; He et al., 2024), we use the LLM to analyze the smaller model's failed attempts. From these attempts, the LLM extracts relevant and correct lemmas, formatted as `have` statements in Lean. This serves to decompose the original problem into more tractable sub-problems for the specialized prover, creating a structured pathway to solve problems that were initially intractable for the prover working alone. A diagram of the method is shown in Figure 2.

We evaluate PROOFCOMPASS on the miniF2F benchmark (Zheng et al., 2021). The results, displayed in Figure 1, highlight the powerful efficiency of our synergistic approach. PROOFCOMPASS achieves a Pass@128 success rate of 55.3%, exceeding the standalone specialized prover's Pass@3200 accuracy of 54.9% while using 25x fewer attempts. The efficiency of our framework is further underscored by its performance on a highly constrained budget: with just 32 attempts, our method's success rate (52.5%) surpasses that of the standalone prover using 4x as many attempts (51.6% at 128 attempts). These results establish that strategic guidance is a powerfully efficient route to state-of-the-art performance. We summarize our contributions as follows:

- We introduce PROOFCOMPASS, a flexible framework that synergistically combines a general-purpose LLM with a specialized prover without requiring additional model training. The core approach is broadly portable across different LLMs and specialized provers.
- We demonstrate that our method achieves superior performance with considerable resource efficiency on the miniF2F benchmark. Our experiments with DeepSeek-Prover-v1.5-RL show that with a 25-fold smaller budget, our method's Pass@128 accuracy (55.3%) is comparable to the standalone prover's performance with Pass@3200 (54.9%), while Pass@32 (52.5%) also outperforms the prover's Pass@128 (51.6%).

2. Related Work

Recent advances in automated theorem proving have demonstrated diverse approaches to combining language models with formal reasoning systems. These approaches can be broadly categorized into specialized models designed specifically for formal mathematics, and large, general-purpose language models adapted for theorem proving. Below, we review key developments in each category.

Specialized Models. Holophrasm (Whalen, 2016) was one of the earliest works that explored training specialized language models — then GRUs — for formal theorem proving, using LMs as policies and value functions in tree search. This foundational effort was later advanced by models like GPT-f (Polu & Sutskever, 2020), which established the viability of using Transformers combined with best-first search. This approach sparked a series of architectural innovations, including Thor (Jiang et al., 2022), which integrated LLMs and symbolic provers, HTPS (Lample et al., 2022), which introduced a novel tree search algorithm for theorem proving, and ReProver (Yang et al., 2023), which used

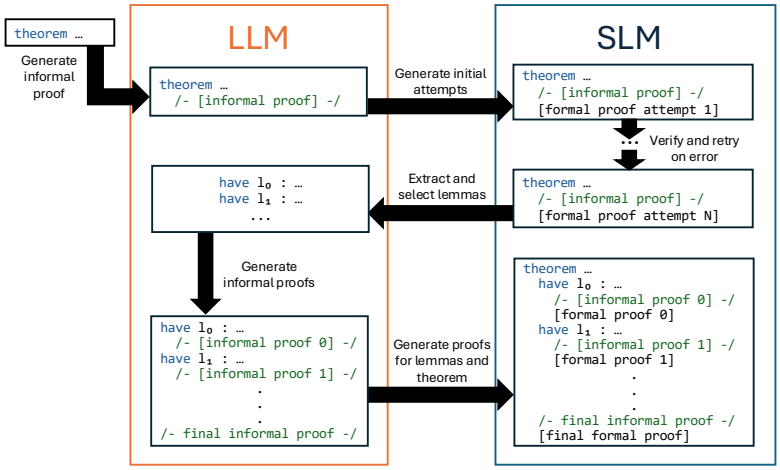

Figure 2. Flowchart of PROOFCOMPASS. A general-purpose Large Language Model (LLM) first provides natural language guidance for a specialized model (SLM, specifically DSP-v1.5-RL) to iteratively generate and verify proof attempts. If all of these attempts prove unsuccessful, the LLM analyzes the failed attempts to select guiding lemmas, generating further informal proofs to direct the SLM in proving these lemmas and the final theorem.

retrieval-augmented generation. More recently, Lean-STaR (Lin et al., 2024) explored training models that interleave informal reasoning in natural language and formal proof steps. POETRY (Wang et al., 2024a) focused on recursive decomposition of proof goals in Isabelle, extracting training data for this decomposition task by restructuring existing human-written proofs. Other models like Goedel-Prover (Lin et al., 2025) and Mathesis-Prover (Xuejun et al., 2025) were trained directly for whole-proof generation, where each pass attempts to generate a complete proof, while InternLM2.5-StepProver (Wu et al., 2024), HunyuanProver (Li et al., 2024), BFS-Prover (Xin et al., 2025), (Lai et al., 2025), and MPS-Prover (Liang et al., 2025) combined specialized models with tree search methods. DeepSeek-Prover-v1.5 (Xin et al., 2024) pushed forward both whole-proof generation and tree search.

General Language Models. Another line of work has focused on leveraging general-purpose language models in theorem proving, hoping to benefit from their substantially better performance in a wide range of reasoning tasks. For instance, COPRA (Thakur et al., 2024) introduced a model-agnostic language agent framework, initially showing promising results with GPT-4 Turbo (OpenAI, 2023). Draft, Sketch and Prove (Jiang et al., 2023) used Minerva (Lewkowycz et al., 2022) to first write a natural language proof and then sketch it as an Isabelle proof with subgoals to be proved by hammers. Other recent work leveraging general-purpose LLMs includes Lyra (Zheng et al., 2024), Subgoal-based prover (Zhao et al., 2023), and LEGO-Prover (Wang et al., 2024b).

Alternatively, automated proof repair frameworks such as ProofAug (Liu et al., 2025) and APOLLO (Ospanov et al., 2025) employ programmatic harnesses with compiler feedback and built-in solvers to correct failed attempts. In contrast, we leverage a second LLM for analyzing failures. Unlike works that use an LLM to directly attempt to fix proofs, like Baldur (First et al., 2023), we use an LLM to extract correct lemmas using the outcomes of failed attempts.

Hybrid Approaches. Recognizing the complementary strengths of specialized and general-purpose models, a promising research direction involves their synergistic combination. For instance, BC-Prover (He et al., 2024) employs an LLM for backward chaining and sub-goal generation, and delegating tactic-level proof steps to specialized provers. In contrast, our method leverages an LLM to analyze the failed proof attempts of a specialized prover to select useful intermediate lemmas, rather than having the LLM directly generate sub-goals.

Specialized large models. Apart from hybrid systems, another powerful yet resource-intensive approach involves directly specializing large models such as Kimina-Prover (72B) (Wang et al., 2025) and DeepSeek-Prover-V2 (671B) (Ren et al., 2025) through further training, a strategy that yields state-of-the-art results but at an often prohibitive cost at most compute budgets. Our work, in contrast, focuses on synergistically combining existing models without incurring these significant training overheads, aiming for a more resource-efficient path to improved performance.

3. Method

In this section, we describe our proposed method for automated theorem proving through the synergistic guidance of a specialized prover by a general-purpose LLM.

3.1. Preliminaries

Our methodology builds upon the DeepSeek-Prover-v1.5-RL (DSP-v1.5) model (Xin et al., 2024), which exhibits a distinct proof generation structure. DSP-v1.5 operates in two primary modes: a Chain-of-Thought (CoT) (Wei et al., 2022) mode and a non-CoT mode. These can be activated using specific prompts, as detailed in (Xin et al., 2024).

The CoT mode initiates the formal proof by generating an informal, natural-language proof sketch (Figure 3). This sketch is embedded within Lean's block comment syntax ('/- ... -/') and is placed immediately after the theorem statement. Following this informal sketch, DSP-v1.5 proceeds to generate the formal Lean 4 code, often interleaving it with short explanatory comments prefixed with '–'. In contrast, the non-CoT mode of DSP-v1.5 generates only Lean 4 code, without any accompanying informal proofs or comments. In this work, we focus on the CoT mode of DSP-v1.5. A common characteristic of DSP-v1.5's proof generations is the production of intermediate steps articulated as 'have' statements within the Lean proof. These 'have' statements effectively function as lemmas or sub-goals, and are a common mechanism for structuring proofs in Lean.

These distinct operational characteristics of DSP-v1.5 inform our approach by suggesting two main avenues for external guidance, which our method aims to exploit:

- **Informal Proof Guidance:** The initial informal proof section generated by DSP-v1.5 in CoT mode provides a natural interface for supplying a high-level, natural language (NL) proof strategy. We leverage this by employing a larger, more capable LLM whose superior reasoning abilities can generate more effective informal proofs than DSP-v1.5 might produce autonomously. This approach is motivated by findings in works such as (Xin et al., 2024) and (Lin et al., 2024), which demonstrated performance improvements by training models on Lean proofs annotated with natural language reasoning steps, thereby showing that NL reasoning can enhance specialized models' performance in formalization tasks. Works like (Jiang et al., 2023; Wang et al., 2024b; Zheng et al., 2024) also use LLM-generated proofs, which they use to guide the general model in its proof generation.

- **Lemma-based Guidance:** The 'have' statements offer a mechanism for decomposing the main theorem and guiding the proof search via structured intermediate goals. An LLM can analyze failed proofs from DSP-v1.5 and strategically select lemmas that effectively partition the problem. Each such lemma then represents a simpler, more constrained subproblem for DSP-v1.5. This strategy of using lemmas or subgoals to augment theorem proving has proven effective in prior systems like BC-Prover (He et al., 2024) and LEGO-Prover (Wang et al., 2024b), which utilize them to structure and simplify the proof search process. Moreover, works such as Lyra (Zheng et al., 2024) and ProofAug (Liu et al., 2025) achieve significant performance gains by repairing failed proofs using automated tools like Sledgehammer (Paulson, 2012) and predefined sets of tactics, thereby demonstrating the potential to extract valuable components from initially unsuccessful derivations.

Our framework's architecture is designed for broad applicability. The guiding LLM is fully interchangeable, allowing the system to immediately benefit from future model improvements, while the core lemma-selection mechanism is adaptable to any specialized prover that generates structured proof attempts. The only component tailored to specific prover features is the injection of informal proof texts, which leverages the CoT-style output of DSP-v1.5 in our experiments. Consequently, PROOFCOMPASS is directly compatible with models that share the exact generation structure of DSP-v1.5, such as Goedel-Prover (Lin et al., 2025). For other provers, the informal proof injection must be modified to suit different CoT structures (Wang et al., 2025) or can be omitted entirely for those lacking CoT support (Xin et al., 2025), with the core lemma-based guidance remaining fully effective in either case.

We now detail how these guidance mechanisms are implemented. Section 3.2 presents our informal proof guidance approach, where the LLM enhances the theorem-proving capabilities of the specialized prover (SLM, DSP-v1.5 in our experiments), while Section 3.3 describes the lemma-based guidance mechanism

3.2. Informal Proof Generation and Initial Attempts

The first stage of PROOFCOMPASS focuses on providing the SLM with a high-quality, natural language (NL) proof to guide its CoT mode.

To achieve this, we initially prompt an LLM with the formal theorem statement T_{formal} and its informal counterpart $T_{informal}$[1] to generate a complete natural language proof,

[1] In miniF2F (Zheng et al., 2021), formal statements are already paired with natural language counterparts. When only a formal statement is available, the LLM itself could be used to *informalize* the statement if needed.

```
theorem algebra_9onxpypzleqsum2onxpy
  (x y z : ℝ)
  (h₀ : 0 < x ∧ 0 < y ∧ 0 < z) :
  9 / (x + y + z) ≤ 2 / (x + y) + 2 / (y + z) + 2 / (z + x) := by
  /-
  To show that for any three positive real numbers \( x \), \( y \), and \( z \), the
  inequality \( \frac{9}{x + y + z} \leq \frac{2}{x + y} + \frac{2}{y + z} + \frac{2}{z + x}
  \) holds, we can use the method of simplifying the expression and applying basic algebraic
  inequalities. Specifically, we will use the fact that the sum of reciprocals of positive
  numbers is greater than or equal to the reciprocal of the sum of those numbers.
  -/
  -- Establish that all sums of pairs of variables are positive.
  have h₁ : 0 < x + y + z := by linarith
  have h₂ : 0 < x + y := by linarith
  have h₃ : 0 < y + z := by linarith
  have h₄ : 0 < z + x := by linarith
  -- Clear the denominators by multiplying both sides by the product of all denominators.
  field_simp [add_pos]
  -- Use the division inequality to compare the numerators.
  rw [div_le_div_iff₀ (by positivity) (by positivity)]
  -- Simplify the expression by expanding and combining like terms.
  ring_nf
  -- Use linear arithmetic to prove the inequality.
  nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x)]
```

Figure 3. Sample proof by DSP-v1.5 of a theorem in Lean 4.

denoted as p_{NL}:

$$(T_{formal}, T_{informal}) \xrightarrow{\text{LLM}} p_{NL} \tag{1}$$

The full NL proof p_{NL} often exceeds the input context capacity of the SLM. Thus, we employ a second LLM query to summarize it, yielding a concise proof summary $p_{summary}$:

$$(T_{formal}, T_{informal}, p_{NL}) \xrightarrow{\text{LLM}} p_{summary} \tag{2}$$

$p_{summary}$ is then embedded within the Lean block comment syntax ('/- ... -/') expected by DSP-v1.5's CoT mode, appended to the formal theorem statement.

With this LLM-generated proof summary ($p_{summary}$) integrated into the input, we query DSP-v1.5 (using its CoT activation prompt) to sequentially generate and verify individual proof attempts. This process continues until either a correct proof is found or the maximum of N initial attempts is reached. Each attempt $i \in \{1, \ldots, N\}$ corresponds to a single generation call:

$$(T_{formal}, p_{NL}) \xrightarrow{\text{SLM}} P^i_{init} \tag{3}$$

Each attempt consumes one unit of the total budget B. If a correct proof is found, the process concludes successfully, and the subsequent lemma extraction and proving stages are skipped. If the N-attempt limit is reached without success, the method proceeds as detailed in the following sections.

3.3. LLM-Guided Lemma Selection

Should the initial N proof attempts fail to prove the theorem, PROOFCOMPASS proceeds to extract and select a set of guiding lemmas. This process begins by aggregating all unique 'have' statements extracted from $P^1_{init}, \ldots, P^N_{init}$ generated by the SLM. This yields a candidate pool of potential lemmas, $L_{extract}$:

$$P^1_{init}, \ldots, P^N_{init} \xrightarrow{\text{lemma extraction}} L_{extract} \tag{4}$$

The candidate pool $L_{extract}$ is first subjected to an automated syntax check to filter out any malformed statements. This initial validation step ensures that only syntactically correct lemmas are passed to the LLM for evaluation, resulting in the set of valid lemmas, L_{valid}:

$$L_{extract} \xrightarrow{\text{syntax checking}} L_{\text{valid}} \tag{5}$$

This set of valid lemmas is then processed by the guiding LLM. Given the pool L_{valid}, the formal theorem statement T_{formal}, its informal counterpart $T_{informal}$, the complete LLM-generated natural language proof p_{NL} for the theorem, and the maximum number of lemmas to select k, the LLM is tasked with selecting at most k lemmas, $L_{select} = (l_0, l_1, \ldots, l_{m-1}), m \leq k$:

$$(L_{valid}, T_{formal}, T_{informal}, p_{NL}, k) \xrightarrow{\text{LLM}} L_{select} \tag{6}$$

To produce L_{select}, the LLM analyzes each candidate lemma in L_{valid}. This involves evaluating the lemma's correctness, primarily considering whether the statement is (i) logically sound and provable using only the global hypotheses of T_{formal}, and (ii) directly justified by p_{NL}. From the subset of lemmas deemed correct, the LLM then performs a strategic selection. This selection aims to identify lemmas covering key inferential steps of p_{NL}, choosing at most k lemmas to constitute the final output set L_{select}, which are ordered according to the logical flow of p_{NL}. The rationale

for this approach of extracting and selecting lemmas from SLM attempts, as opposed to direct LLM generation, is detailed in Appendix C.

In the second stage, the lemmas in L_{select} are passed to the LLM for informal proof generation. The LLM generates a concise informal proof p_l for each selected formal lemma $l \in L_{select}$, derived from p_{NL}, T_{formal} and $T_{informal}$. It also generates a concise informal proof for the main theorem, p_{main}, which assumes the truth of all selected lemmas in L_{select}:

$$(l, T_{formal}, T_{informal}, p_{NL}) \xrightarrow{\text{LLM}} p_l \tag{7}$$

$$(T_{formal}, T_{informal}, L_{select}, p_{NL}) \xrightarrow{\text{LLM}} p_{main} \tag{8}$$

The LLM adapts explanations from p_{NL} to generate a proof for each lemma l. The collection of all these generated informal proofs is denoted by $\mathcal{P} = \{p_l \mid l \in L_{select}\} \cup \{p_{main}\}$.

Next, L_{proven} is initialized by extracting lemma proofs from previous attempts. For each $l \in L_{select}$, we search $P^1_{init}, \ldots, P^N_{init}$ for all its occurrences as a 'have' statement. For each such occurrence with an associated proof $P_{l,extract}$, we verify if $P_{l,extract}$ proves l using only the global hypotheses of T_{formal}. If this check passes and l is not already in L_{proven}, we add l and that $P_{l,extract}$ to L_{proven}:

$$(P^1_{init}, \ldots, P^N_{init}, T_{formal}, L_{select}) \rightarrow L_{proven} \tag{9}$$

The outcome of this stage is thus the curated set of formal lemmas L_{select}, the set of their corresponding LLM-generated informal proofs $\mathcal{P}$, and the initialized set of proven lemmas L_{proven}. These are then used to guide subsequent formal proof attempts by DSP-v1.5, as detailed in the next section.

3.4. Lemma Proving and Theorem Validation

This stage proceeds to prove the theorem T_{formal} using the set of selected lemmas L_{select} and their corresponding informal proofs $\mathcal{P}$. In case no lemmas were selected, the method reverts to the initial proof generation strategy described in Section 3.2 for the remainder of the total budget B.

Throughout this stage, each call to the SLM to generate P_{main} or any individual lemma proof P_{l_i} consumes an attempt from the overall budget B allocated for the problem. If the attempts allocated for this lemma-based strategy are exhausted before a complete and verified proof P_{final} is obtained, this approach is considered unsuccessful for the current theorem.

Initially, the method seeks to obtain P_{main}, a formal proof of the main theorem T_{formal} under the assumption that all lemmas in L_{select} hold. The SLM is prompted to generate P_{main} using T_{formal} (with all lemmas in L_{select} appended as hypotheses) and the LLM-generated informal proof for the main theorem p_{main}:

$$(T_{formal}, L_{select}, p_{main}) \xrightarrow{\text{SLM}} P_{main} \tag{10}$$

This P_{main} provides the concluding logic to derive T_{formal} from the established lemmas.

If P_{main} is successfully generated, an iterative procedure begins. Within each iteration of this loop, the method first attempts to prove individual lemmas. It iterates through each lemma $l_i \in L_{select}$ (for i from 0 to $m-1$). If l_i is not already in L_{proven}, l_i is formulated as a distinct Lean 4 theorem. This theorem includes all global hypotheses of T_{formal} and all previously ordered lemmas from L_{select} that precede l_i as hypotheses, with the goal being the formal statement of l_i. The SLM then attempts to generate a formal proof P_{l_i} for this lemma, guided by l_i's informal proof p_{l_i}:

$$(l_0, ..., l_i, T_{formal}, p_{l_i}) \xrightarrow{\text{SLM}} P_{l_i} \tag{11}$$

If P_{l_i} is successfully generated and verified by Lean 4, l_i is added to L_{proven}.

Following this lemma-proving pass within an iteration, a candidate full proof, P_{final}, is constructed for T_{formal}. This P_{final} starts with T_{formal} and its global hypotheses, incorporates 'have' statements for each lemma $l_k \in L_{proven}$ along with their formal proofs P_{l_k}, and finally appends the proof steps from the previously generated P_{main}. This complete P_{final} is then submitted to Lean 4 for validation. If P_{final} is a valid proof of T_{formal}, the theorem is considered proven, and the process concludes successfully.

4. Experiment

In this section we describe the experimental evaluation of our proposed method.

4.1. Experimental Setup

Dataset. We evaluate our approach on the widely used miniF2F dataset (Zheng et al., 2021). It contains 488 formal mathematics problems, consisting of olympiad-level problems (AMC, AIME, IMO) and undergraduate mathematics exercises formalized in various proof assistant systems, including Isabelle (Paulson, 1994), HOL Light (Harrison, 2009), Metamath (Megill & Wheeler, 2019), and Lean (de Moura & Ullrich, 2021). The Lean problems were originally in Lean 3, but were converted to Lean 4 in (Yang, 2023).

Evaluation. We assess the performance of PROOFCOMPASS using the Pass@k metric, specifically reporting

Table 1. Pass@k performance on miniF2F-test

METHOD	BUDGET	MINIF2F-TEST
BASELINES		
BC-PROVER	100	30.7%
BC-PROVER + REPROVER	100	31.6%
BC-PROVER + LLMSTEP	100	32.0%
LEGO-PROVER	100	50.0%
LYRA	200	51.2%
DSP-V1.5	32	$50\% \pm 0.5\%$
	128	$51.6\% \pm 0.5\%$
	3200	$54.9\% \pm 0.7\%$
OURS		
PROOFCOMPASS	32	52.5%
	128	**55.3%**
ABLATIONS		
ONLY LEMMA GUIDANCE	32	51.6%
	128	53.3%
ONLY INFORMAL PROOF	32	52.0%
	128	52.9%

Pass@32 and Pass@128 due to computational budget constraints. For our methodology, an "attempt" in the Pass@k calculation corresponds to any call made to the DSP-v1.5 model. This includes initial proof generation attempts, as well as individual attempts to prove each selected lemma and the theorem itself using the lemmas (see Appendix A for a detailed breakdown of LLM task and SLM attempt timings, including overhead considerations). The validity of generated formal proofs and selected lemmas is verified by interacting with Lean 4 via the Lean REPL (leanprover-community, 2023). A proof attempt is deemed successful if the Lean REPL returns no error messages and the generated proof does not contain any `sorry` placeholders. Conversely, an attempt is considered a failure if it produces errors, includes `sorry` placeholders, or if the Lean verification process exceeds a 20-second timeout.

Baselines. Our primary point of comparison is the performance of the DSP-v1.5 model as reported in its original publication (Xin et al., 2024). To contextualize our method's performance, we also compare against BC-Prover (He et al., 2024), a relevant LLM-driven framework for theorem proving in Lean (considered both as a standalone LLM-based prover and in its hybrid configurations with ReProver (Yang et al., 2023) and LLMStep (Welleck & Saha, 2023)), and other prominent LLM-based provers such as LEGO-Prover (Wang et al., 2024b) and Lyra (Zheng et al., 2024), all of which utilize the Isabelle proof assistant. This selection of baselines reflects our study's central aim: to introduce and evaluate a novel LLM-guidance methodology for a given specialized prover, using DSP-v1.5 as our specific test case. We therefore do not include comparisons with other standalone specialized provers, as our primary goal is to assess the LLM-guidance technique itself, rather than to conduct a wide-ranging benchmark of all prover types.

Implementation Details. Our methodology utilizes specific Large Language Models for its guidance components. The initial, comprehensive natural language proof (p_{NL}) for a given theorem is generated using Gemini 2.0 Flash Thinking Experimental (Kane, 2025). All other LLM-driven tasks within our method, such as summarizing p_{NL}, performing lemma analysis and selection, and elaborating the proofs for each lemma, are performed using Gemini 2.0 Flash (Pichai et al., 2024). For all interactions with the DeepSeek-Prover-v1.5-RL model, we employ its Chain-of-Thought (CoT) mode, using the specific prompt provided in (Xin et al., 2024). All attempts were generated using an Nvidia GeForce RTX 4090, and verification was conducted using Lean version 4.15.

Hyperparameters. As we compare our method with the Pass@32 and Pass@128 performances of DSP-v1.5, the total generation budget is $B = 128$. The initial number of attempts was chosen to be $N = 16$, as this often resolves a significant portion of problems upfront by DSP-v1.5 and provides a sufficient corpus of 'have' statements from failed attempts for effective LLM guidance, while selecting at most $k = 5$ lemmas focuses on key strategic steps. Appendix B elaborates on these choices and includes supporting analysis.

4.2. Main Results

The performance of our proposed method on the miniF2F-test dataset is presented in Table 1, with a visual comparison to the DeepSeek-Prover-v1.5-RL baseline for Pass@32 and Pass@128 rates depicted in Figure 1. Our approach demonstrates notable improvements in theorem-proving capabilities and resource efficiency, particularly when enhancing its foundational specialized prover, DSP-v1.5.

When compared directly to this baseline, our method yields substantial gains. As shown in Table 1, our Pass@32 rate of 52.5% exceeds the baseline's Pass@128 rate of 51.6%. More strikingly, with a 25-fold smaller computational budget, our method's Pass@128 rate of 55.3% matches the average 54.9% performance that DSP-v1.5 achieves with 3200 attempts.

Further contextualizing these results, our method also shows strong performance relative to other contemporary systems. A key contribution of this work is its novel hybrid architecture. To the best of our knowledge, BC-Prover (He et al., 2024) is the most comparable contemporary hybrid approach; our method (55.3% Pass@128) significantly outperforms BC-Prover + LLMStep (32.0% Pass@100), underscoring the efficacy of our specific synergistic design.

Moreover, even with a limited budget of 32 attempts, our method's Pass@32 rate of 52.5% is competitive with or exceeds other LLM-based methods using $3-6\times$ larger attempt budgets, such as LEGO-Prover (50.0% at 100 attempts) and Lyra (51.2% at 200 attempts).

These combined results underscore the significant advantage of our synergistic approach, achieving strong performance and resource efficiency by effectively leveraging the complementary strengths of general-purpose LLMs and specialized provers.

4.3. Ablation Study

To better understand the contributions of the primary components of our methodology (LLM-generated informal proof guidance and LLM-guided lemma selection) we conducted two ablation studies. The results are included in Table 1.

The first ablation, "Only Lemma Guidance," isolates the impact of the lemma-based guidance mechanism. In this configuration, the initial LLM-generated informal proof $p_{summary}$, subsequent informal proofs for lemmas p_{l_i} and the main theorem p_{main} are omitted. Instead, DSP-v1.5 generates its own informal proofs (as in its standard CoT mode, illustrated in Figure 3), while still benefiting from the LLM's analysis of failed attempts to extract and select guiding 'have' statements (lemmas). This ablation proved highly effective on its own; in particular, its Pass@32 performance of 51.6% already matches the baseline's Pass@128 result, and its total Pass@128 rate reached 53.3%. This suggests that strategically decomposing the problem using LLM-selected lemmas already aids the specialized prover, even without direct informal proof guidance from the LLM for each step. However, a high-quality informal proof still provides an extra performance boost.

The second ablation, "Only Informal Proof," evaluates the effectiveness of providing LLM-generated informal proofs alone without the subsequent lemma processing. Here, the LLM generates the initial informal proof $p_{summary}$, which guides DSP-v1.5 for all $N = 128$ initial attempts (as described in Section 3.2), but the lemma extraction, selection, and proving stages (Sections 3.3 and 3.4) are entirely skipped. This configuration yielded a Pass@32 of 52.0% and a Pass@128 of 52.9%, which also outperforms the baseline. Notably, its Pass@32 rate surpasses the baseline's Pass@128 rate (51.6%), demonstrating that leveraging the LLM's advanced reasoning for high-level proof strategies enables the specialized prover to find correct formalizations with significantly fewer attempts by effectively narrowing the search space.

Together, these results highlight that both the LLM-generated informal proof guidance and the LLM-driven lemma processing contribute to the overall performance of our method. Their contributions are complementary: their combination yields the best performance. PROOFCOMPASS thus demonstrates the benefits of a synergistic interaction between the LLM and the specialized prover.

5. Conclusion

In this paper, we introduced PROOFCOMPASS, a novel methodology to enhance automated theorem proving by synergistically leveraging the high-level reasoning of general-purpose Large Language Models (LLMs) to guide specialized provers. Our approach employs an LLM to generate strategic natural language proofs and to analyze failed proof attempts by a specialized model to extract and select useful intermediate lemmas, thereby decomposing complex problems.

The empirical validation on the miniF2F (Zheng et al., 2021) benchmark highlights the framework's remarkable resource efficiency. PROOFCOMPASS is on par with the 3200-attempt average performance of the standalone prover while using only 128 attempts—a 25-fold computational saving. This efficiency extends to smaller scales, where our Pass@32 performance surpasses the baseline's Pass@128 accuracy. This work thus establishes that intelligently orchestrating the complementary strengths of LLMs and specialized provers offers a more promising and resource-efficient path towards more capable theorem proving systems.

6. Limitations and Future Work

The proposed methodology, while demonstrating promising results, presents several limitations that also outline avenues for future research.

While our core lemma selection loop is designed for broad applicability to any prover generating structured proof attempts, the informal proof guidance component in its current form is tailored to the specific CoT-style input of provers like DSP-v1.5. A key avenue for future work is therefore to both validate the core lemma-based framework across a more diverse range of specialized provers and to develop more generalized techniques for the informal guidance component.

Furthermore, the experimental evaluation presented herein was conducted with a maximum of 2^7 attempts per problem. This represents a considerably more constrained computational budget compared to several contemporary works in automated theorem proving, which often report Pass@k metrics frequently exceeding 2^{17} attempts (Li et al., 2024; Wu et al., 2024; Xin et al., 2024). Consequently, an important avenue for future research involves assessing the performance scaling of our methodology under significantly increased computational resources.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Finance Code 001. The authors also wish to thank Professor Adriano Côrtes for his valuable assistance with the computational experiments in this work.

Impact Statement

This work presents a resource-efficient methodology for automated theorem proving. By demonstrating a path forward that does not require large-scale computational infrastructure, our approach helps to democratize research in this domain. We believe this focus on computational efficiency is a positive contribution that can foster broader and more inclusive participation within the machine learning community.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Avigad, J. Mathematics and the formal turn. *Bulletin of the American Mathematical Society*, 61(2):225–240, 2024.

de Moura, L. and Ullrich, S. The lean 4 theorem prover and programming language. In Platzer, A. and Sutcliffe, G. (eds.), *Automated Deduction – CADE 28*, pp. 625–635, Cham, 2021. Springer International Publishing.

First, E., Rabe, M. N., Ringer, T., and Brun, Y. 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*, pp. 1229–1241, 2023.

Fraser, M., Granville, A., Harris, M. H., McLarty, C., Riehl, E., and Venkatesh, A. Will machines change mathematics? *Bulletin (New Series) of the American Mathematical Society*, 61(2), 2024.

Harrison, J. HOL Light: An overview. In Berghofer, S., Nipkow, T., Urban, C., and Wenzel, M. (eds.), *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pp. 60–66, Munich, Germany, 2009. Springer-Verlag.

He, Y., Zhang, J., Bao, J., Lin, F., Yang, C., Qin, B., Xu, R., and Yin, W. Bc-prover: Backward chaining prover for formal theorem proving. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3059–3077, 2024.

Jiang, A. Q., Li, W., Tworkowski, S., Czechowski, K., Odrzygóźdź, T., Miłoś, P., Wu, Y., and Jamnik, M. Thor: Wielding hammers to integrate language models and automated theorem provers. *Advances in Neural Information Processing Systems*, 35:8360–8373, 2022.

Jiang, A. Q., Welleck, S., Zhou, J. P., Lacroix, T., Liu, J., Li, W., Jamnik, M., Lample, G., and Wu, Y. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, 2023.

Kane, P. Access the latest 2.0 experimental models in the gemini app., February 2025. URL https://blog.google/feed/gemini-app-experimental-models.

Lai, J., Zhang, J., Xu, S., Chen, T., Wang, Z., Yang, Y., Zhang, J., Cao, C., and Xu, J. Llm-based automated theorem proving hinges on scalable synthetic data generation. *arXiv preprint arXiv:2505.12031*, 2025.

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

leanprover-community. A simple repl for lean 4. https://github.com/leanprover-community/repl, 2023. Accessed: May 20, 2025.

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Li, Y., Du, D., Song, L., Li, C., Wang, W., Yang, T., and Mi, H. Hunyuanprover: A scalable data synthesis framework and guided tree search for automated theorem proving. *arXiv preprint arXiv:2412.20735*, 2024.

Liang, Z., Song, L., Li, Y., Yang, T., Zhang, F., Mi, H., and Yu, D. Mps-prover: Advancing stepwise theorem proving by multi-perspective search and data curation. *arXiv preprint arXiv:2505.10962*, 2025.

Lin, H., Sun, Z., Welleck, S., and Yang, Y. Lean-star: Learning to interleave thinking and proving. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024.

Lin, Y., Tang, S., Lyu, B., Wu, J., Lin, H., Yang, K., Li, J., Xia, M., Chen, D., Arora, S., et al. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025.

Liu, H., Sun, J., Li, Z., and Yao, A. C. Efficient neural theorem proving via fine-grained proof structure analysis. *arXiv preprint arXiv:2501.18310*, 2025.

Megill, N. D. and Wheeler, D. A. *Metamath: A Computer Language for Mathematical Proofs*. Lulu Press, Morrisville, North Carolina, 2019. http://us.metamath.org/downloads/metamath.pdf.

OpenAI. GPT-4 Turbo, 2023. URL https://platform.openai.com/docs/models/gpt-4-turbo.

Ospanov, A., Farnia, F., and Yousefzadeh, R. Apollo: Automated llm and lean collaboration for advanced formal reasoning. *arXiv preprint arXiv:2505.05758*, 2025.

Paulson, L. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Schmidt, R. A., Schulz, S., and Konev, B. (eds.), *PAAR-2010: Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning*, volume 9 of *EPiC Series in Computing*, pp. 1–10. EasyChair, 2012. doi: 10.29007/tnfd. URL /publications/paper/Mzp.

Paulson, L. C. *Isabelle: A Generic Theorem Prover*. Springer Verlag, 1994.

Pichai, S., Hassabis, D., and Kavukcuoglu, K. Introducing gemini 2.0: our new ai model for the agentic era, December 2024. URL https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024.

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

Ren, Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Thakur, A., Tsoukalas, G., Wen, Y., Xin, J., and Chaudhuri, S. An in-context learning agent for formal theorem-proving. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=V7HRrxXUhN.

Venkatesh, A. Some thoughts on automation and mathematical research. *Bull. Amer. Math. Soc.(NS)*, 61(2):203–210, 2024.

Wang, H., Xin, H., Liu, Z., Li, W., Huang, Y., Lu, J., Zhicheng, Y., Tang, J., Yin, J., Li, Z., et al. Proving theorems recursively. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a.

Wang, H., Xin, H., Zheng, C., Liu, Z., Cao, Q., Huang, Y., Xiong, J., Shi, H., Xie, E., Yin, J., et al. Lego-prover: Neural theorem proving with growing libraries. In *The Twelfth International Conference on Learning Representations*, 2024b.

Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., Santos, M. D., Sung, F., Vinyes, M., Ying, Z., Zhu, Z., et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Welleck, S. and Saha, R. llmstep: Llm proofstep suggestions in lean. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*, 2023.

Whalen, D. Holophrasm: a neural automated theorem prover for higher-order logic. *arXiv preprint arXiv:1608.02644*, 2016.

Wu, Z., Huang, S., Zhou, Z., Ying, H., Wang, J., Lin, D., and Chen, K. Internlm2. 5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 2024.

Xin, H., Ren, Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., et al. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024.

Xin, R., Xi, C., Yang, J., Chen, F., Wu, H., Xiao, X., Sun, Y., Zheng, S., and Shen, K. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. *arXiv preprint arXiv:2502.03438*, 2025.

Xuejun, Y., Zhong, J., Feng, Z., Zhai, P., Yousefzadeh, R., Ng, W. C., Liu, H., Shou, Z., Xiong, J., Zhou, Y., et al. Mathesis: Towards formal theorem proving from natural languages. *arXiv preprint arXiv:2506.07047*, 2025.

Yang, K. minif2f-lean4. https://github.com/yangky11/miniF2F-lean4, 2023.

Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R. J., and Anandkumar, A. Leandojo: Theorem proving with retrieval-augmented

language models. *Advances in Neural Information Processing Systems*, 36:21573–21612, 2023.

Yang, K., Poesia, G., He, J., Li, W., Lauter, K., Chaudhuri, S., and Song, D. Formal mathematical reasoning: A new frontier in ai. *arXiv preprint arXiv:2412.16075*, 2024.

Zhao, X., Li, W., and Kong, L. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. *arXiv preprint arXiv:2305.16366*, 2023.

Zheng, C., Wang, H., Xie, E., Liu, Z., Sun, J., Xin, H., Shen, J., Li, Z., and Li, Y. Lyra: Orchestrating dual correction in automated theorem proving. *Transactions on Machine Learning Research*, 2024.

Zheng, K., Han, J. M., and Polu, S. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021. Available at: https://github.com/openai/miniF2F.

Supplementary material for

PROOFCOMPASS: Enhancing Specialized Provers with LLM Guidance

This supplementary document is organized as follows:

- **Section A** presents a computational time analysis for each component of the proposed workflow.
- **Section B** provides justification for key hyperparameter choices.
- **Section C** compares the chosen strategy of extracting lemmas from the specialized prover's failed attempts with direct LLM generation.
- **Section D** provides the complete prompts used for all LLM tasks to ensure reproducibility.

A. Generation Times

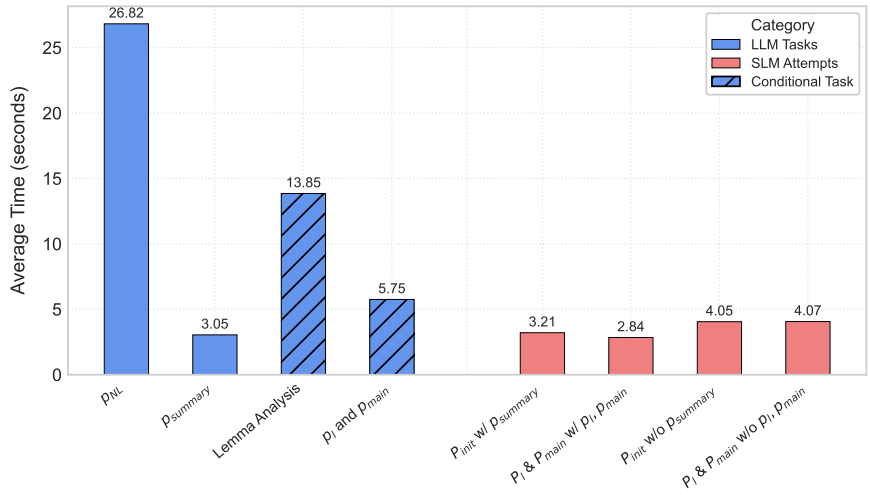

Figure 4. A detailed breakdown of the average time, in seconds, for each component of the PROOFCOMPASS workflow compared to the baseline. The chart categorizes operations into three types: one-time LLM guidance tasks (solid blue), conditional LLM tasks for lemma processing (striped blue), and per-attempt costs for the Specialized Language Model (SLM) (red).

This section details the computational timings of individual LLM guidance tasks and Specialized Language Model (SLM) proof attempts, followed by an analysis of their impact on overall time-to-solution performance.

Figure 4 presents the average durations for key operations. Initial LLM tasks, such as p_{NL} generation and subsequent summarization, constitute an upfront time investment, with specific durations detailed in the figure. Lemma analysis also introduces an overhead if a problem is not solved within N = 16 initial attempts. A crucial benefit of this LLM pre-processing is evident in the SLM attempt times: proof attempts guided by an LLM-generated $p_{summary}$ are notably faster (average 3.21s) than the baseline DSP-v1.5 CoT attempts (average 4.05s). This speed-up occurs because the baseline model, in its CoT mode, must first generate its own informal proof sketch, as illustrated in Figure 3, whereas our method provides this sketch as input. Similarly, the generation of informal proofs for lemmas (p_l) and the main theorem (p_{main}), while adding to LLM task time, leads to more efficient SLM generation for the corresponding lemma and final theorem proofs.

A quantitative analysis reveals when the upfront time cost of our method is offset by its greater per-attempt efficiency. For problems requiring more than $N = 16$ attempts, PROOFCOMPASS incurs a significant fixed time overhead of approximately 100.83 seconds. This accounts for initial LLM guidance tasks (29.87s), the first 16 failed SLM attempts (51.36s), and the conditional lemma processing (19.60s). Following this, each subsequent attempt to prove a lemma or the main theorem is highly efficient, costing only 2.84 seconds. In contrast, the baseline method costs 4.05 seconds per attempt, regardless of

the stage. Consequently, the baseline's cumulative time surpasses PROOFCOMPASS's full workflow after approximately 46 total attempts. This indicates that while the baseline may be faster for problems solved in fewer than 46 attempts, PROOFCOMPASS's approach becomes more time-efficient for more complex problems requiring a larger computational budget.

B. Hyperparameters Analysis

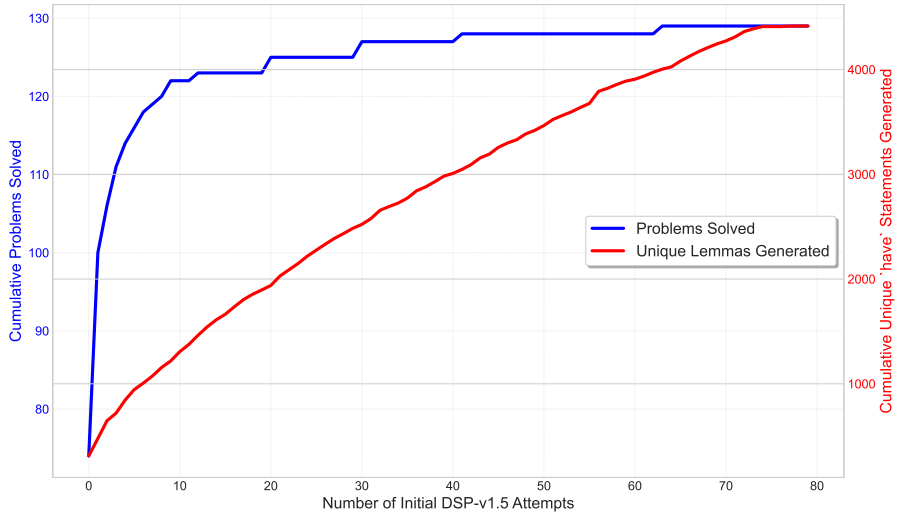

Figure 5. Impact of the number of initial DSP-v1.5 attempts (N) on the cumulative number of problems solved (blue line, left y-axis) and the cumulative number of unique 'have' statements (potential lemmas) generated (red line, right y-axis). This data was collected from the initial proof generation stage (Section 3.2), where DSP-v1.5 was guided by the LLM-generated proof summary, prior to any subsequent lemma extraction (Section 3.3) or proving (Section 3.4) cycles.

This section provides the justification for the key hyperparameters used in the experiments (Section 4). The number of initial proof attempts by DSP-v1.5 was set to $N = 16$. This choice is informed by the trends observed in Figure 5. While the rate of solving new problems (blue curve) significantly diminishes after approximately $N = 10 - 15$ attempts, the pool of unique 'have' statements (potential lemmas, red curve) continues to expand, reaching around 2000 unique statements close to $N = 20$. Setting $N = 16$ thus aims to capture a majority of the problems solvable by DSP-v1.5 in its initial attempts while still generating a diverse corpus of material from failed attempts for effective LLM-guided lemma extraction (Section 3.3).

Regarding the maximum number of selected lemmas, this was set to $k = 5$. This value was established through preliminary qualitative evaluations, which indicated that up to five strategically chosen lemmas were generally adequate to cover the essential inferential steps of most proofs. Our methodology also allows the LLM to select fewer than k lemmas if a smaller set is deemed optimal. Requesting a larger k in these evaluations risked the inclusion of trivial or redundant lemmas, which could dilute the specialized prover's focus and inefficiently expend its attempt budget on less critical sub-problems. Therefore, $k = 5$ encourages the LLM to identify a concise yet impactful set of guiding lemmas for DSP-v1.5.

C. Alternative Lemma Generation Strategies

An alternative to our adopted method of extracting and selecting lemmas from the Specialized Language Model's (SLM, DSP-v1.5) attempts is to directly prompt a Large Language Model (LLM) to generate intermediate lemmas. This section elaborates on why the former approach was chosen. The primary reason, identified in early experimentation, is that lemmas directly generated by an LLM, while often semantically correct, tended to be syntactically different from those typically produced by DSP-v1.5 and proved more challenging for the SLM to utilize.

For instance, in the problem `numbertheory_x5neqy2p4`, directly LLM-generated lemmas often express constraints using set theory (e.g., stating a variable is an element of a set of possible values), as can be seen in Figure 7. In contrast, lemmas extracted from SLM attempts (Figure 6) frequently articulate equivalent constraints as disjunctions of direct equality

```
theorem numbertheory_x5neqy2p4
  (x y : ℤ) :
  x^5 ≠ y^2 + 4 := by
  have l₀ : x^5 % 11 = 0 ∨ x^5 % 11 = 1 ∨ x^5 % 11 = 10 :=
by
    sorry
  have l₁ : (y^2 + 4) % 11 = 2 ∨ (y^2 + 4) % 11 = 4 ∨ (y^2 +
4) % 11 = 5 ∨ (y^2 + 4) % 11 = 7 ∨ (y^2 + 4) % 11 = 8 ∨
(y^2 + 4) % 11 = 9 := by
    sorry
  have l₂ : x^5 % 11 ≠ (y^2 + 4) % 11 := by
    sorry
  sorry
```

Figure 6. Lemmas extracted from failed SLM attempts.

```
theorem numbertheory_x5neqy2p4
  (x y : ℤ) :
  x^5 ≠ y^2 + 4 := by
  have l₀ : ∀ (x : ℤ), x^5 % 11 ∈ ({0, 1, 11} : Set ℤ) := by
    sorry
  have l₁ : ∀ (y : ℤ), (y^2 + 4) % 11 ∈ ({2, 4, 5, 7, 8, 9}
: Set ℤ) := by
    sorry
  have l₂ : Disjoint ({0, 1, 11} : Set ℤ) ({2, 4, 5, 7, 8, 9}
: Set ℤ) := by
    sorry
  sorry
```

Figure 7. Lemmas generated by the LLM

assertions for the variable. While semantically equivalent, these syntactic variations significantly impacted the SLM's proving efficiency. In a comparative experiment where the SLM was tasked with proving theorems using these two styles of guiding lemmas (requiring proofs for each lemma P_{l_i} and the main theorem P_{main}), the approach using LLM-extracted, SLM-style lemmas required an average of 12 attempts. Conversely, using directly LLM-generated lemmas required an average of 17.25 attempts (both averaged over 4 independent runs), a difference of over 40%.

This empirical difference suggests that the SLM is more adept at processing lemmas that conform to syntactic patterns closer to its own typical outputs or training data. Therefore, our methodology prioritizes extracting and selecting lemmas from the SLM's own attempts. This aim is to provide strategically sound intermediate steps in a form that is more tractable for the SLM, thereby optimizing the overall resource efficiency of the theorem-proving process.

D. LLM Prompts

This appendix provides all prompts used in our experiments for reproducibility.

D.1. User Prompt for Natural Language Proof Generation

> Provide a proof in natural language for the theorem below:
> Theorem statement in Lean 4:
> <formal_statement>
> Informal Statement:
> <informal_statement>
> Please provide a rigorous, detailed proof using natural language.

D.2. System Prompt for Proof Summarization

> You are a mathematical proof summarizer. You will be given a formal statement of a theorem in Lean 4, its informal counterpart, and a natural language proof of the theorem. Your response should contain only a summarized version of the natural language proof that's both concise and complete. You should start you response with "We want to show that", "We have", "We need to show that" or "To show that". You should write only the proof in your response, nothing more.

D.3. User Prompt for Proof Summarization

THEOREM STATEMENT IN LEAN 4:
<formal_statement>
INFORMAL STATEMENT:
<informal_statement>
COMPLETE PROOF:
<nl_proof>
Write your summarized proof below

D.4. System Prompt for Lemma Selection

You are a theorem-proving assistant trained to evaluate individual statements for their provability as standalone lemmas in Lean 4. Your task is to analyze a single list of candidate lemma statements, and then select at most 5 to cover all key steps of the natural language proof.

Core Principle for Lemma Evaluation

A candidate lemma statement L is "correct" if and only if a Lean 4 statement `have [name] : [statement] := by [proof]` would be successfully provable using ONLY the global assumptions/hypotheses provided in the formal theorem statement.

A lemma is "incorrect" if it couldn't be proven in Lean 4 using only the assumptions and hypothesis provided in the, or if it's not fully justified by the natural language proof based on the global assumptions.

Specific Criteria Guiding "Correct" vs. "Incorrect" Evaluation

1. **Global Provability:** The **exact mathematical statement** of the lemma must be provably true using only the given hypotheses in the formal theorem statement.
2. **Justification by NL Proof:** The lemma must be a direct and logical step or assertion found in the natural language proof, understandable from the global context.
3. **No Dependence on Undischarged Assumptions:**
   - If the proof proceeds by cases (e.g., "Case 1: Assume P... then R", "Case 2: Assume Q... then T"), a lemma stating P by itself, Q by itself, R by itself (if R depends on P), or T by itself (if T depends on Q) is **incorrect**. These statements are only true under temporary, local assumptions.
   - However, a lemma stating $P \vee Q$ (if this disjunction is provable globally) would be **correct**.
   - Similarly, $P \to R$ or $Q \to T$ might be **correct** if these implications are what the proof establishes.
4. **Contradictions & Unjustified Steps:**
   - If a lemma contradicts a statement in the proof or a hypothesis, it's **incorrect**.
   - If a lemma makes an assertion not present or derivable from the NL proof and global hypotheses, it's **incorrect**.
   - In a proof by contradiction, if 'A' is true and the proof temporarily assumes 'not A' to reach a contradiction, a lemma stating 'not A' as a factual step from the global context is **incorrect**.
5. **Consequences of Incorrect Lemmas:** If a lemma B follows logically from another lemma A, and lemma A is determined to be "incorrect", then lemma B is also "incorrect".

Input Format You'll receive the formal statement of the theorem in Lean 4, its informal counterpart, a complete natural language proof, and a list of candidate lemma statements.

THEOREM STATEMENT IN LEAN 4:
<Formal statement in Lean 4>
INFORMAL STATEMENT:
<Informal translation of theorem statement>

COMPLETE PROOF:
<Complete natural language proof of theorem>
LEMMAS:
0: <lemma in Lean 4>
1: <lemma in Lean 4>
...
N-1: <lemma in Lean 4>
Output Format (Strictly Adhere to This Structure) LEMMA ANALYSIS
0:
Analysis: [Provide a short, step-by-step analysis of lemma 0. Explain precisely why it IS or IS NOT provable as a standalone 'have' statement given ONLY the theorem's global hypotheses and the NL proof. Reference the **Core Principle** and **Specific Criteria** above.]
Evaluation: [correct or incorrect]
1:
Analysis: [Analysis for lemma 1 as above.]
Evaluation: [correct or incorrect]
...
N-1:
Analysis: [Analysis for lemma N-1 as above.]
Evaluation: [correct or incorrect]
SELECTION RATIONALE FOR CHOSEN LEMMAS
AVAILABLE LEMMAS: [List all lemmas that you evaluated as 'correct'.]
REASONING: [Based on your 'Evaluation' of all original lemmas above, select up to 5 correct lemmas that represent the key intermediate steps from the provided natural language proof. Do not select lemmas that are restatements of the theorem's hypotheses or its final conclusion. Always try to select 5 lemmas, choosing fewer only if there are not enough correct intermediate lemmas available. The final chosen lemmas must be ordered to reflect the logical flow of the natural language proof.]
CHOSEN LEMMAS

[Write all chosen lemmas here. Each chosen lemma must be presented in the following format. Number them sequentially starting from l_0, regardless of their original index.]

have l_0 : <statement> := by
have l_1 : <statement> := by
...
This is how your output should look like
LEMMA ANALYSIS
0:
Analysis: [Analysis of lemma 0 based on the criteria explained above]
Evaluation: [Evaluation of lemma 1 (correct or incorrect)]
1:
Analysis: [Analysis of lemma 0 based on the criteria explained above]
Evaluation: [Evaluation of lemma 1 (correct or incorrect)] ...
N-1:
Analysis: [Analysis of lemma 0 based on the criteria explained above]
Evaluation: [Evaluation of lemma 1 (correct or incorrect)]
SELECTION RATIONALE FOR CHOSEN LEMMAS
AVAILABLE LEMMAS:
[first correct lemma statement]
...
[last correct lemma statement]
REASONING: [Reasoning about which (AT MOST 5) lemmas to choose, with the goal of selecting the key

intermediate steps of the NL proof, excluding hypotheses and the conclusion, and ordering them according to the proof's logical flow.]
CHOSEN LEMMAS:
[List of chosen lemmas. Note that the subscripts should be in the order 0, 1, 2, ..., irrespective of the number of its corresponding lemma in the input.]

have l_0 : <statement> := by
have l_1 : <statement> := by

D.5. User Prompt for Lemma Selection

THEOREM STATEMENT IN LEAN 4:
<formal_statement>
INFORMAL STATEMENT:
<informal_statement>
COMPLETE PROOF:
<nl_proof>
LEMMAS:
0: <lemma 0>
1: <lemma 1>
...
N-1: <lemma N-1>
Proceed with your response below, strictly following all guidelines and output structures specified in the system prompt.

D.6. System Prompt for Lemma and Final Proof Generation

You are a mathematical proof analyst. Your task is to analyze a given mathematical proof (provided alongside formal and informal statements) and break it down into a sequence of logical steps, formatted clearly for potential formalization in Lean 4.
You will receive:

1. A formal theorem statement in Lean 4, potentially including hypothesis labels like h_0, h_1.
2. An informal statement of the theorem.
3. A complete natural language proof of the theorem.
4. Formal lemma statements (e.g., **have** $l_0 : < ... >$) that will appear in the Lean proof.

Analyze the provided natural language proof and follow these guidelines meticulously to structure your output:

1. **REASONING Section:**
   - Start with **REASONING:**.
   - Analyze the provided natural language proof to identify its overall strategy and key insights.
   - Explicitly note how the formal lemma statements (**have** l_0, **have** l_1, etc.) map to the key steps in the natural language proof. These will directly correspond 1-to-1 with the l_0 :, l_1 : steps you generate. You must NEVER include new lemma statements that are not present in the input.
   - Briefly list the main steps you identified in the provided proof, ensuring each step aligns with one of the provided formal lemma statements. Remember to never include new lemma statements in the steps.

- In the case the formal lemma statements don't match the steps in the proof well, you should try to modify the natural language proof to make it match the formal lemma statements. You should NEVER modify the formal lemma statements themselves, or add new lemma statements.

2. **STEPS Section:**
   - Follow with **STEPS:**.
   - Label the key milestones as l_0 :, l_1 :, etc. These must exactly match the order and content of the formal **have** statements provided.
   - Each step (l_i :) must state a precise mathematical fact in natural mathematical language.
3. **Step Proofs (Proof: sections):**
   - Follow each l_i : statement with **Proof:**.
   - Provide a concise summary of the justification found in the provided natural language proof.
   - To show dependency on a previous step l_j, state the mathematical result of l_j *as a fact* within the narrative summary of the proof for l_i, instead of mentioning the name l_j directly. You should NEVER mention the name l_j inside the proof.
   - Correct: "Since <mathematical statement from l_j holds / was established>, the proof proceeds by..."
   - Incorrect: "By l_j..." or "Using the result from l_j...". This is incorrect because 'l_j' is present in the proof, which is not allowed.
4. **Final Proof Section:**
   - End with **Final Proof:**.
   - Summarize how the proof combines the results stated in the l_i steps to reach the final conclusion.
   - State intermediate results directly as established facts, without referencing step labels.
5. **Formatting:**

   Your response must follow this exact structure:

   REASONING:
   <Analysis noting correspondence between formal lemmas and steps>

   STEPS:
   l_0 :
   <First mathematical statement matching first **have**>
   Proof:
   <Detailed explanation with all necessary calculations>

   l_1 :
   <Next mathematical statement matching next **have**>
   Proof:
   <Detailed explanation with all necessary calculations>

   ...

   Final Proof:
   <Conclusion using established results>

Here's an example showing the expected output format:
<example from miniF2F-valid>

D.7. User Prompt for Lemma and Final Proof Generation

THEOREM STATEMENT IN LEAN 4:
<formal_statement>
INFORMAL STATEMENT:
<informal_statement>
COMPLETE PROOF:
<nl_proof>
FORMAL LEMMAS::
<lemmas>