A Minimal Proof Language for Neural Theorem Proving over Isabelle/HOL

***Abstract*—Neural Theorem Proving (NTP) employs deep learning methods, particularly Large Language Models (LLMs), to automate formal proofs in proof assistants. This approach holds promise for reducing the dramatic labor costs or computation costs required in proof engineering, which is fundamental to formal verification and other software engineering methods. The paper explores the potential of improving NTP by redesigning the proof language, given that LLMs' capabilities depend highly on representations. We introduce *MiniLang*, a redesigned proof language for Isabelle/HOL incorporating an improved version of Sledgehammer. Experiments show MiniLang benefits two fine-tuned LLMs by improving the success rate on the PISA benchmark by up to 29% in comparison to generation of Isar proof script. The success rate under one attempt (so-called *pass@1*) reaches 69.1%, exceeding the previous Baldur's pass@64 (65.7%); The pass@8 reaches 79.2%, exceeding the state-of-the-art on PISA (71.0%) achieved by Magnushammer.**

I. Introduction

Formal verification exhaustively examines software systems for specific vulnerabilities and certifies crucial safety and correctness properties. This process serves as an essential mechanism for ensuring program safety in safety-critical scenarios. Theorem proving—whether interactive or automated—represents a key component of formal verification and often becomes the primary bottleneck in the process.

Interactive theorem proving (ITP) relies on *proof assistants*, specialized software that enables users to construct certified proofs. Automated theorem proving (ATP) proves goals automatically at the cost of expressiveness. Both of them face challenges in formal verification: ITP requires substantial manual effort when verifying large systems; ATP may fail if the automation is not powerful enough.

The rise of LLMs has the potential to change this status quo. So-called Neural Theorem Proving (NTP) uses LLMs to interface with interactive proof assistants, enabling automated proof construction for complex properties without sacrificing expressiveness [1]–[4]. Promisingly, recent NTP can prove a substantial number of math competition problems, including those at the International Mathematical Olympiad level, achieving near 90% success on miniF2F [3].

Unfortunately, recent NTP works seem to focus primarily on LLMs' mathematical reasoning capabilities, with extensive attention given to math competitions (miniF2F [5], PutnamBench [6], and [7]–[10]), while benchmarks (PISA [11], CoqGym [12], and [13]–[20]) derived from the wider body of real-world proof engineering appear to have been less prioritized. Based on our brief survey [21] of 46 arXiv articles on NTP uploaded since 2024, 30 works focus exclusively on math competitions, only 7 consider general mathemantical formalisations, and 9 consider broader applications of theorem proving common to computer science such as system security, program verification, or computation theory.

Among existing NTP works, approaches fall into two sorts — search-based approach and declarative approach — according to the role of the theorem prover's *tactic language* in the approaches.

A tactic is a subroutine that either solves proof goals directly or transforms them into a simpler form. In tactic-based proofs, a proof is constructed through the composition of tactics, represented as a sequence of tactics, e.g., fig. 1a. Tactics are direct computational transformations of the proof state that are generally difficult to understand intuitively. Consequently, tactic-based proofs make explicit the computational process of soundly transforming a proof's premises into its goals, rather than conveying the intuitive ideas of the proof as typically presented in textbooks.

In the field of NTP, the search-based approach essentially uses LLMs to provide heuristics that guide the search for a sequence of tactics that proves the goal. The declarative approach, by contrast, trains LLMs to generate proof outlines (e.g., fig. 1c) that decompose large proof goals into smaller subgoals. The ability to express declarative proofs is heavily dependent on the capabilities of the chosen underlying *theorem prover*. Isabelle [22] is a theorem prover which supports particularly expressive declarative proofs where users write structural proof outlines, and can delegate the discharge of subgoals to ATPs such as Sledgehammer [23] and SMT solvers (fig. 1c). Manual tactics are employed only when ATPs fail. Consequently, the role of tactics is minimized, leaving clean proof outlines that express the proof idea. Lean, a theorem prover which has seen particular popularity in NTP, finds itself in an intermediate position between these two styles (fig. 1b). While it provides syntax for writing in a declarative style, tactics still play a key role in Lean because generic automation tools [24], [25] like hammers [26] are not yet ready or stable. Coq is a popular theorem prover with the lowest level of automation among those discussed here — its proofs are almost exclusively tactic-based.

Prior work suggests that a maximally declarative approach offers the most promising path to NTP. LLMs excel at proposing high-level intuitions and proof outlines [3], [31], but struggle with detailed stepwise logical inference [32]–[34] creating challenges in producing high-quality tactic sequences without extensive search and attempts. Allowing the LLM to focus on high-level proof structure is complementary to the strengths

```
Theorem sqrt2_not_rational :
  ∀p q : nat, q <> 0 → p * p = 2 * (q * q) → ⊥.
intros p q; generalize p; clear p;
elim q using (well_founded_ind lt_wf).
clear q; intros q Hrec p Hneq;
pose proof Hneq as Hlt_O_q;
apply Nat.neq_0_lt_0 in Hlt_O_q;
intros Heq.
apply (Hrec (3 * q - 2 * p)
         (comparison4 _ _ Hlt_O_q Heq)
         (3 * p - 4 * q)).
apply sym_not_equal; apply lt_neq;
apply Nat.add_lt_mono_l with (2 * p);
rewrite ←plus_n_O; rewrite Nat.add_comm;
rewrite Nat.sub_add; auto with *.
apply new_equality; auto.
Qed.
```

(a) A tactic-based proof (Coq, from [27]). Tactics are highlighted.

```
theorem sqrt_two_irrational
  {a b : ℕ} (co : gcd a b = 1) : a^2 ≠ 2 * b^2 :=
by rintro h : a^2 = 2 * b^2
  have : 2 | a^2 := by simp [h]
  have : 2 | a := dvd_of_dvd_pow prime_two this
  apply Exists.elim this rintro c aeq
  have : 2 * (2 * c^2) = 2 * b^2 := by
    simp [Eq.symm h, aeq]
    simp [pow_succ' _, mul_comm, ...]
  have : 2 * c^2 = b^2 := by
    apply mul_left_cancel₀ _ this decide
  have : 2 | b^2 := by simp [Eq.symm this]
  have : 2 | b := by
    exact dvd_of_dvd_pow prime_two this
  have : 2 | gcd a b := by
    apply dvd_gcd . assumption . assumption
  have _ : 2 | (1 : ℕ) := by simp [co] at *
  contradiction
```

(b) A mixture of tactics and declarations, (Lean, from [28]).

```
theorem sqrt2_not_rational: "sqrt 2 ∉ ℚ"
proof
  let ?x = "sqrt 2"
  assume "?x ∈ ℚ"
  then obtain m n :: nat where
    "|?x| = m / n" and "coprime m n" by ATP
  hence "m^2 = ?x^2 * n^2" by ATP
  hence eq: "m^2 = 2 * n^2"
    using of_nat_eq_iff power2_eq_square by ATP
  hence "2 dvd m^2" by ATP
  hence "2 dvd m" by ATP
  have "2 dvd n" proof -
    from "2 dvd m" obtain k where "m = 2 * k" by ATP
    with eq have "2 * n^2 = 2^2 * k^2" by ATP
    hence "2 dvd n^2" by ATP
    thus "2 dvd n" by ATP
  qed
  with "2 dvd m" have "2 dvd gcd m n" by ATP
  with lowest_terms have "2 dvd 1" by ATP
  thus False using odd_one by ATP
qed
```

(c) A declarative proof (Isar in Thor [29] style, modified from [30])

```
theorem sqrt2_not_rational: "sqrt 2 ∉ ℚ"
RULE
INTRO
LET ?x = "sqrt 2"
CONSIDER m n :: nat where
    A1: "|?x| = m / n" and A2: "coprime m n" END
HAVE B: "m^2 = ?x^2 * n^2" END WITH A1 A2
HAVE eq: "m^2 = 2 * n^2"
  END WITH B of_nat_eq_iff power2_eq_square
HAVE C: "2 dvd m^2" END WITH eq
HAVE D: "2 dvd m" END WITH C
HAVE E: "2 dvd n"
  CONSIDER k where F: "m = 2 * k" END WITH D
  HAVE G: "2 * n^2 = 2^2 * k^2" END WITH F eq
  HAVE H: "2 dvd n^2" END WITH G
END WITH H
HAVE I: "2 dvd gcd m n" END WITH E
HAVE J: "2 dvd 1" END WITH I lowest_terms
END WITH odd_one J
```

(d) MiniLang translation of the Isar proof

Fig. 1: Several proof languages. $(a \mid b)$ and $(a \ \mathtt{dvd}\ b)$ denote integer division of a by b.

of ATPs, which can then work to discharge local subgoals. Moreover in a setting such as Lean where declarative and tactics-based proofs are commonly mixed, the LLM must learn both styles of proof in more detail, increasing the complexity of the learning problem.

The declarative approach, however, depends on three critical supports from the underlying proof assistant: a declarative proof language, powerful proof automation that minimizes reliance on tactics, and a large training corpus written in the declarative style.

Isabelle best fulfills these requirements in real-world proof engineering. Beyond its declarative language and powerful automation, a key differentiator lies in the training corpus. (1) *Scale*: Isabelle's standard library [35] and Archive of Formal Proofs (AFP) [36] provide ∼5.4M lines of code and ∼356K proofs. By contrast, Lean's corpus stems mainly from mathlib [37], containing ∼1.2M lines; the latest Coq corpus from CoqStop [38] collects ∼197K proofs. (2) *Style*: Isabelle's corpus focuses on declarative proofs, while Coq's relies on tactic-based proofs unsuitable for declarative NTP training. (3) *Scope*: Isabelle's corpus spans diverse areas across program verification, computer security, computation theory, and math. By contrast, Lean's corpus focuses almost entirely on math, which cannot represent the full scope of proof engineering.

However, these advantages have not led to flourishing development of declarative NTP on Isabelle. Only 8 out of the recent 58 arXiv papers [21] are based on Isabelle. On the other hand, researchers have turned to Lean, where they have had to settle for impure declarative approaches that interweave tactics due to Lean's limited automation infrastructure; they have also expended significant resources building and expanding declarative-style training corpora. Even these imperfect methods yield excellent results [3], [4].

Evidently, something impedes the development of declarative NTP in Isabelle, thereby hindering progress across the entire field, given that we believe Isabelle is the optimal foundation for declarative proof in real-world proof engineering.

We discussed these issues with key figures in the Isabelle community and NTP researchers [3], [39], [40] who used to work on Isabelle but have turned to Lean. Their feedback indicates two major issues for NTP using Isabelle: 1) A concurrent Read-Eval-Print-Loop (REPL) infrastructure for machine learning is absent; 2). Isabelle's proof language, *Isar*, while highly declarative, is designed to emulate mathematicians' relaxed writing and is unfriendly for LLMs to learn. An article [41] reports 76% of failures in their system when attempting to produce Isar are classified as syntax errors. A detailed analysis of Isar's deficiencies in NTP contexts is provided in § II.

These motivate us to develop a REPL infrastructure for clusters, and more importantly, mitigate Isar's deficiencies in NTP by designing a new proof language, *MiniLang*. An example is illustrated in fig. 1d. MiniLang eliminates unnecessary features designed for human usability and readability, aiming to distill the essential elements of declarative proofs (§ III).

However, as a new language, no corpus exists for MiniLang, making it challenging to train LLMs on it. To overcome this, a rule-based translator from Isar to MiniLang is built (§ IV). Using it, we translate $\sim$85% of AFP into MiniLang, obtaining a large corpus of real-world proof engineering. Consequently, we are able to evaluate MiniLang by fine-tuning NTP models over MiniLang and Isar, to compare the two (§ V).

The result shows that MiniLang brings up to 29% increment on PISA, a benchmark consisting of $\sim$3K randomly selected proof goals from AFP, in comparison to generation of Isar alone. We summarize the paper's contributions as follows.

- We explore the impact of proof languages on the performance of NTP for the first time, with a specific focus on real-world proof engineering goals.
- We present MiniLang, the first proof language specifically designed for declarative NTP. It mitigates Isar's human-friendly but machine-unfriendly features.
- We develop a translator from Isar to MiniLang that successfully converts 85.28% of AFP proofs.
- We present a socket-based Isabelle REPL infrastructure, capable of cluster usage.

Though this research is based on Isabelle, our contributions extend beyond any single proof assistant. By answering the following research questions, this work offers a novel direction for improving NTP via proof language redesign (RQ1, RQ2), and experience in training NTP with custom languages (RQ3).

RQ1: Can we effectively improve the declarative NTP by redesigning the underlying proof language?
Yes. Models on MiniLang supersede the models on Isar by 29% with pass@1 on PISA, while the training corpus of MiniLang stems from an $\sim$85% portion of Isar's.

RQ2: Is improvement confined to syntax error reduction or are reasoning errors also reduced
MiniLang reduces not only syntax errors but also **reasoning errors**, with the performance gains extending well beyond mere syntactic correctness.

RQ3: For a freshly crafted proof language, how can we obtain effective training corpora for this language?
Rule-based translation is an approach, which is effective even when the translation is incomplete ($\sim$15% entries fail to be translated in our case).

II. Issues in Isar

Our journey begins with analyzing Isar's potential issues that impose unnecessary learning burdens on LLMs.

A. Mathematician-Friendly but Machine-Unfriendly

Isar is a human-readable language specially designed for presenting proofs in a style resembling mathematicians' pen-and-paper writing. While we believe the declarative style of Isar is appropriate for NTP, specific design choices of Isar are not optimised for this purpose.

When chaining proof steps, Isar prefers connectives and context-sensitive indirect references while avoiding direct use of names. Examples include connectives (**from**, **with**, **then**, **thus**, **hence**, **also**, **moreover**, **ultimately**, **finally**) and pronouns (`this`, `that`). While this makes proof scripts flow more like human writing, it also introduces ambiguities. The exact references of these pronouns depend on the contextual connectives used. Subtly incorrect uses of these connectives can cause unexpected behaviors in critical proof operations like case split and induction, leading to proof failures. Worse still, there is no local syntactic way to correct such errors, as the semantics of these connectives are closely related.

Isar even exploits English ambiguity to hide important technical distinctions from readers. Examples include (`also`, `finally`) vs. (`moreover`, `ultimately`). In English, these pairs are virtually indistinguishable. However, in Isar, they have subtly different behaviours. (`moreover`, `ultimately`) collect lemmas as arguments for later ATPs, while (`also`, `finally`) apply transitive calculation over the collected lemmas (e.g., to infer $a_1 \leq a_n$ from $a_1 \leq a_2, \cdots, a_{n-1} \leq a_n$). It is possible that these unnecessary complexities create difficulties for representation-based LLMs, especially when these keywords appear interchangeably in the pretraining data.

Conversely, it is common in Isar for there to be multiple redundant ways to express the same proof procedure. Examples include fig. 2a which allows a forward writing starting from an assumption A, stepwisely establishing a series of lemmas $\{B_i\}_n$, and finally reaching the conclusion C. Except for the writing flow, fig. 2a is behaviorally identical to the more common fig. 2b.

Redundant syntax and syntactic sugar might be harmless to humans. However, the situation differs for NTP due to corpus scarcity. When training data for one syntactic form is insufficient, redundant syntax either dilutes the already limited corpus or requires LLMs to expend additional effort learning that different representations are semantically equivalent.

```
{
  fix x
  assume A
  have B₁ by t₁ ...
  have Bₙ by tₙ
  have C by u
}
```

(a) One possible way

```
have C if A for x
proof -
  have B₁ by t₁ ...
  have Bₙ by tₙ
  show C by u
qed
```

(b) A more common way

Fig. 2: Two redundant ways of introducing lemmas in Isar

B. Legacy burdens on Isabelle's proof languages

Isar is neither the unique nor the original proof language of Isabelle. Before Isar, Isabelle used a backward tactic language resembling Coq, e.g., **subgoal for** x **apply** tac_1 **apply** tac_2 **done**. Isar's design is self-contained and capable of fully replacing the old language. However, the obsolete language has been preserved for compatibility reasons. As a result, the old language still appears in the AFP corpus, and some authors mix both languages when they find legacy features more convenient. This preservation has exacerbated syntax redundancy issues. An example is **proof** tac $\equiv$ **apply** tac **proof** –, which causes `proof` to provide redundant functionality that overlaps with `apply`, violating the principle of minimality.

III. MiniLang: A Minimal Proof Language

Isar is designed for generic logics, human-readable presentations, and convenient use for human experts. Many of these design choices introduce redundant elements that complicate LLMs' learning. This motivates us to propose *MiniLang*, a minimalist proof language for Isabelle/HOL that eliminates unnecessary syntax elements and language constructs, aiming to distil declarative proofs to their essential form.

A. Syntax

Both Isar and MiniLang structure proof scripts as sequences of statements, where each statement begins with a keyword called a *command*.

$$\begin{aligned}\text{Proof-Script} &::= (\text{Statement})^{+}\\ \text{Statement} &::= \text{Command}\ (\text{Argument})^{*}\end{aligned}$$

When Isar contains over 29 commands, MiniLang minimizes this number to 5 core commands + 8 extensions for common operations + 3 for the compatibility with Isar. Table I summarizes these commands, while their formal syntax is defined in our supplementary Reference Manual.

B. The Proof Model and Semantics

Both Isar and MiniLang are imperative in the sense that every statement manipulates an underlying state machine (fig. 3) that represents proof states and their transitions.

Isar's state machine consists of three modes (forward, backward, chaining) and hierarchies of nested sub-proofs, making it relatively complex and difficult to formalize. By contrast, MiniLang employs a simpler state machine where each state is represented as a labeled tree organizing subgoals into contexts:

$$\begin{aligned}\text{Tree} &::= \text{Leaf} \mid (\textit{label:}\ \text{Context}, \textit{children:}\ \text{Tree}^{+})\\ \text{Context} &::= (\text{a set of variables}, \text{a set of named hypotheses})\\ \text{Leaf} &::= \text{Context} \vdash \text{Goal} \qquad \text{Goal} ::= \text{Term}\end{aligned}$$

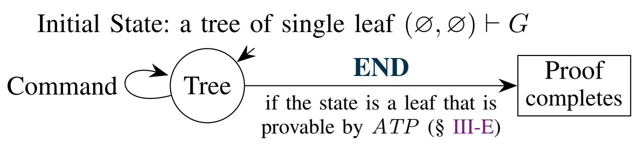
Leaves represent unproven subgoals that require proofs. Each non-leaf node groups related subgoals that share a common context, typically arising from the decomposition of a larger proof goal. The resulting tree provides an organized hierarchy of outstanding proof obligations at a given state. For example, the resulting tree after executing the statement **CONSIDER** `k` **where** `"m = 2 * k"` at line 13 of fig. 1d is

$$\begin{tikzcd}[column sep=tiny]
 & & (\{m,n\},\{\sqrt{2}\in\mathbb{Q},\mathtt{A1},\mathtt{A2},\mathtt{B},\mathtt{C},\mathtt{D}\}) \arrow[dl, dash] \arrow[dr, dash] & \\
 & (\varnothing,\varnothing) \arrow[dl, dash] \arrow[dr, dash] & & (\varnothing,\{2\ \mathrm{dvd}\ m\}) \vdash \mathrm{False} \\
(\varnothing,\varnothing) \vdash \exists k.\, m = 2k & & (\{k\},\{m = 2k\}) \vdash 2\ \mathrm{dvd}\ m &
\end{tikzcd}$$

The tree contains three subgoals arranged left to right. The leftmost subgoal (opened by line 13) requires proving the existence of k. The middle subgoal, opened by **HAVE** E at line 12, is the parent of the first subgoal. Its context has variable k fixed with condition $m = 2k$, allowing it to use this condition once the first subgoal establishes k's existence. Similarly, the rightmost subgoal (False) is the top goal of the entire proof.

TABLE I: MiniLang's Commands.

Command	Description
Core Commands for Declarative Proofs	
INTRO	moves all hypotheses of the proof goal and fixes all ∀-quantified variables into the context.
HAVE	introduces intermediate lemmas.
CONSIDER	fixes variables subject to certain conditions and/or analyzes the proof goal by cases.
END	ends the proof of a subgoal or the top goal.
NEXT	ends the proof of a subgoal and turns to the next sibling subgoal.
Extensions for Common Proof Operations	
RULE	deduces the goal by a given rule or a default rule chosen by Isabelle. A rule specifies how to derive a conclusion or how to destruct a hypothesis.
SIMPLIFY	rewrites the goal using Isabelle's system simplifier.
UNFOLD	unfolds user-specified definitions.
CHOOSE x	instantiates leading ∃ in the goal by a witness x.
CASE_SPLIT	applies structural case analysis to the goal.
INDUCT	applies induction to the goal.
LET	defines abbreviations of terms.
NOTATION	defines syntactical notations.
Auxiliary Commands for Compatibility with Isar Proofs	
CONFIG	configures Isar attribute system.
OPEN	opens either Isar bundles or Isar modules (locales).
APPLY	applies arbitrary tactics as users want, designed for compatibility only and not recommended to use because tactics undermine declarative proofs.



Initial State: a tree of single leaf $(\varnothing,\varnothing)\vdash G$
Command
Tree
END
if the state is a leaf that is provable by ATP (§ III-E)
Proof completes

Fig. 3: MiniLang's state machine for proving proposition G.

It can utilize the conclusion $(2 \text{ dvd } m)$ from the middle subgoal. Each leaf subgoal's context includes all labeled contexts from its ancestors in the tree, so all subgoals can access the previously established lemmas `A1`, `A2`, `B`, `C`, `D`.

The initial state is the tree of a single root node that represents the top goal itself, e.g.,

$$\Big((\varnothing,\varnothing)\vdash \sqrt{2}\notin\mathbb{Q}\Big) \tag{1}$$

The semantics of every MiniLang statement is defined as a transition that replaces either the leftmost leaf or the leftmost non-leaf node of the tree of a state. For example, the transition of the **RULE** at line 2 replaces the leftmost leaf (i.e. the root node (1) in this example),

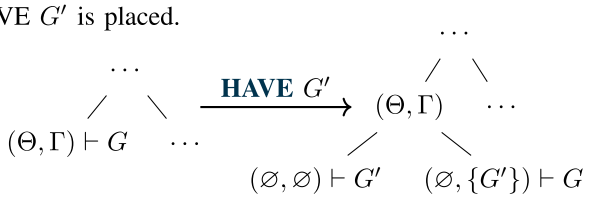
$$\Big((\varnothing,\varnothing)\vdash \sqrt{2}\notin\mathbb{Q}\Big) \xrightarrow{\textbf{RULE}} \Big((\varnothing,\varnothing)\vdash \sqrt{2}\in\mathbb{Q}\longrightarrow \text{False}\Big)$$

It reduces the goal by applying the rule of contradiction (which is the default rule of $\notin$, as configured in Isabelle).

The next **INTRO** resembles Coq's `intros`. It moves all hypotheses in the goal into the context and binds them to generated fresh names. It also fixes all $\forall$-quantified variables into the context. Generally for any variable set Θ, hypothesis set Γ, variables $\{x_i\}_m$, and propositions $\{H_i\}_n, G$,

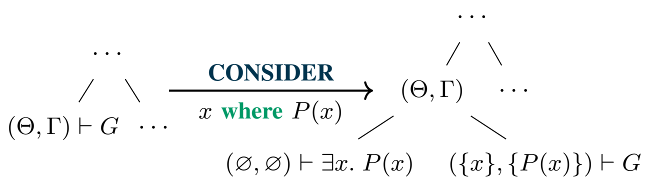
$$(\Theta,\Gamma)\vdash \forall x_1,\cdots,x_m.\, H_1\longrightarrow\cdots\longrightarrow H_n\longrightarrow G$$
$$\xrightarrow{\textbf{INTRO}} (\Theta\cup\{x_i\}_m, \Gamma\cup\{name_i : H_i\}_n)\vdash G$$

HAVE and CONSIDER are the key to declarative proofs. HAVE G' introduces a subgoal G' so that the conclusion of the goal can be used in proving the contextual goal G in which HAVE G' is placed.



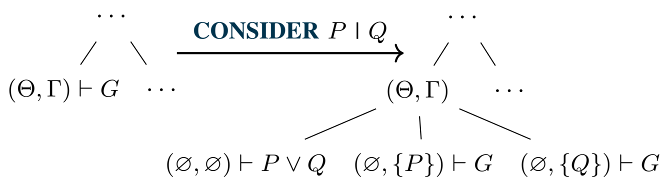
$\cdots$
$(\Theta,\Gamma)\vdash G$
HAVE G'
(Θ,Γ)
$(\varnothing,\varnothing)\vdash G'$
$(\varnothing,\{G'\})\vdash G$

CONSIDER merges Isar's `obtain` and `consider` (as both commands perform elimination of disjunctive connectives). It then has two usages. **CONSIDER** x **where** $P(x)$ fixes a variable x subject to $P(x)$ if such variable exists.



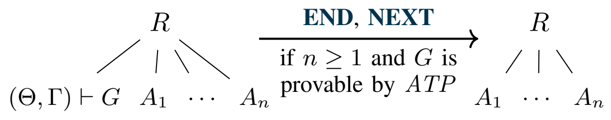
$\cdots$
$(\Theta,\Gamma)\vdash G$
CONSIDER
x where $P(x)$
(Θ,Γ)
$(\varnothing,\varnothing)\vdash \exists x.\ P(x)$
$(\{x\},\{P(x)\})\vdash G$

CONSIDER can also split a goal by cases. As an instance, **CONSIDER** $x > 0 \mid x = 0 \mid x < 0$ divides the proof into three cases, when x is positive, is zero, or is negative.



$\cdots$
$(\Theta,\Gamma)\vdash G$
CONSIDER $P \mid Q$
(Θ,Γ)
$(\varnothing,\varnothing)\vdash P\vee Q$
$(\varnothing,\{P\})\vdash G$
$(\varnothing,\{Q\})\vdash G$

HAVE and CONSIDER provide sufficient structural constructs for declarative proofs, allowing complex proof goals to be recursively decomposed into subgoals simple enough for an ATP. These subgoals are then closed by statement END or NEXT, which invokes the ATP to prove the subgoals.



R
$(\Theta,\Gamma)\vdash G$
A_1
$\cdots$
A_n
END, NEXT
if $n\geq 1$ and G is provable by ATP

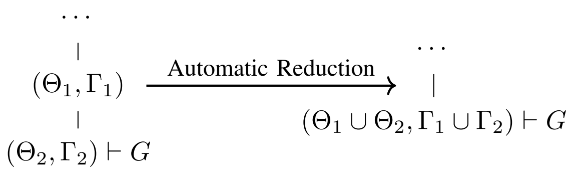
(2)

END and NEXT actually have the same semantics. The two versions are introduced just for the sake of readability.

The ATP used by END and NEXT is parameterized. Our implementation uses Sledgehammer* (§ III-E). If the ATP fails to prove the goal, an error is raised.

END and NEXT are the only ways to conclude a subgoal. Even when another command like APPLY is capable of proving a subgoal, MiniLang intentionally leaves a placeholder $(\Theta,\Gamma)\vdash$ True that must be explicitly closed by END or NEXT. This ensures well-structured MiniLang proofs.

The transition diagram (2) requires $n \geq 1$, but it is still complete because MiniLang ensures that any non-leaf node in the tree of any state must have two or more children. This invariant is preserved through the following automatic reduction rule that is applied whenever possible:



$\cdots$
(Θ_1,Γ_1)
$(\Theta_2,\Gamma_2)\vdash G$
Automatic Reduction
$(\Theta_1\cup\Theta_2,\Gamma_1\cup\Gamma_2)\vdash G$

Finally, a special use of **END** is to conclude the final top goal at last, e.g., line 19 in fig. 1d. The entire proof process in MiniLang thus consists of using commands to operate the state machine, to transform the proof state until the proof is completed with this final END, as illustrated in fig. 3.

C. *Other Commands and Mechanism*

In addition to the structural proof commands, MiniLang provides commands for common proof operations like induction and structural case analysis (analysis over constructors of a term). The semantics of these commands are defined in the Reference Manual attached to our supplementary materials.

Admittedly, these commands can be viewed as a certain kind of tactic. However, they are designed to have a general semantics that mirror standard operations in pen-and-paper proofs. Therefore, the presence of these commands does not undermine the alignment between the formal proofs and pen-and-paper proofs. Thus, we consider these commands relatively harmless to declarative proofs.

MiniLang also introduces a transparent automatic calculation mechanism to eliminate the need for the calculation keywords **also** and **finally** (cf. § II) in most cases. Isar's calculation is about extending (order-theoretical) chains, e.g., using $A_n \leq A_{n+1}$ to extend a chain $A_1 \leq \cdots \leq A_n$ to derive $A_1 \leq A_{n+1}$. MiniLang maintains a series of chains internally. Once a proposition P is proven, MiniLang checks if P can extend any existing chain at its tail. If so, all feasible chains are extended; otherwise, P initiates a new chain of length 1 in the internal series. Once a chain $\{A_i\}_n$ reaches a length of 2 or more, indicating a derivation has occurred, the calculation result $A_1 \leq A_n$ is added into the context or updated if it is already there. This allows users to reference the calculated result without explicit uses of `also` and `finally`. Though not complete, this approach can replace many uses of `also` and `finally`, reducing the concepts involved in MiniLang.

D. Soundness

Because MiniLang's proof operations are ultimately defined in terms of Isabelle's existing kernel operations and SMT interfaces, these present no new threats to soundness in comparison to Isar.

E. Sledgehammer* : An Improved Sledgehammer

MiniLang uses an improved version of Sledgehammer as its ATP backend to verify declarative proofs.

The core function of Sledgehammer is *premise selection* — selecting from a database the relevant lemmas that can possibly help to prove a given goal. These lemmas are later sent to SMT solvers to finally obtain a proof of the given goal.

This premise selection is based on heuristics [42] and classical machine learning methods like k-NN [43]. MiniLang improves this by providing syntax that allows LLMs to suggest relevant lemmas and identify those that should be avoided:

END / NEXT WITH *relevant-lemmas* **WITHOUT** *avoided-lemmas*

The lemmas are hints that Sledgehammer* preferentially (but not mandatorily) considers for use or avoidance. If a lemma is not found in the context (i.e., undefined reference), it is simply ignored as it is merely a hint. This mitigates problems when LLMs generate non-existent lemmas due to hallucinations.

We also improve on Sledgehammer's integration with Isabelle's simplification system. First, tactic `auto` is applied before invoking Sledgehammer, which rewrites the goal using Isabelle's system simplifier. If `auto` timeouts, a safer tactic `clarsimp` is used instead. Second, we use a simplification-based brute-force tactic `fastforce` in parallel to Sledgehammer as an additional ATP backend. Moreover, the annotated lemmas in the **WITH** clause are also sent to the tactics. However, these tactics require the lemmas to be labelled by their intended usage, such as rewrite rules or introduction rules used for sequent calculus. We build a heuristic to guess the usage of a given lemma from its name and expression. This exploits the full power of the tactics.

IV. Translation from Isar to MiniLang

Training NTP models requires substantial proof data, which does not exist for our newly designed language. We address this through automated translation from Isar to MiniLang, successfully converting 85.28% of AFP's proofs, obtaining ∼285K proofs from AFP's ∼324K proofs.

Moreover, this translation process highlights three key strategies that embody our minimalist principle for redesigning a proof language to enhance the performance of NTP:

- **Elaboration**: Make implicit information explicit by exposing hidden details in a clear, structured manner;
- **Normalization**: Consolidate diverse approaches for achieving the same logical purpose;
- **Elimination of tactics**: Eliminate tactics except for canonical ones that indicate key proof steps and mirror common operations in pen-and-paper proofs.

A. The Translation Process

The translation process includes 20+ passes, each of which represents one step from an intermediate language to another closer to MiniLang.

1) Parse text into Abstract Syntax Tree (AST).
2) Elaboration & Normalization
   - Unfolding sugars and macro variables like `?thesis` and `?case`. Add type annotations to variables.
   - Assign anonymous lemmas with generated names.
   - Resolve pronouns `this` and `that`.
   - Normalize all connectives into **using** and reference lemmas by explicit names, e.g. **have** A ... **then have** B **by** t ≡ **have** A ... **have** B **using** A **by** t
   - Isar provides two ways to reference lemmas, by name or by expression of the proposition. This step replaces expression-based references with name references.
   - Rewrite the obsolete **subgoal** statement into the usual **proof-qed** structure.
   - Tactics can occur in **proof**, **qed**, **apply**, **by**. We normalize the various ways of applying tactics into **apply**.
   - Definition unfolding can be invoked either by tactic `unfold_tac` or by command unfolding. We normalize them to `unfold_tac`.
   - **proof-qed** blocks can contain multiple **next**-separated sub-blocks, each of which may contain multiple **show** statements, and each statement may target multiple goals. These elements can appear in arbitrary orders. We normalize the structure so that each **proof** block targets exactly one goal, and all blocks are ordered consistently with their corresponding goal occurrences.
   - Isar variables and hypotheses are *selectively* fixed into the context in arbitrary order. We normalize this by requiring *all* variables and hypotheses to be declared at the proof body beginning in their occurrence order.
   - Tactics `induct_tac`, `induct` and `case_tac` are subtle variants of `induction` and `case`. This step normalizes the former into the latter.

- Normalize tactic `goal_cases` into the standard **proof** - **case** - **qed** structure.
- Normalize bracket structure (e.g., fig. 2a) into **proof-qed** structure (e.g., fig. 2b).
- Tactics can be combined by combinators (`, + ? | [`n`]`). We eliminate combinators and normalize tactic applications into sequences of atomic tactics.
- One tactic can affect multiple goals (e.g., `auto` affects all goals). We restrict such tactics and duplicate their applications as needed to ensure that each application targets only the leading goal.

3) Translating into MiniLang. By this stage, most redundant Isar statements have already been eliminated. The remaining normalized statements are translated into their corresponding MiniLang equivalents: **obtain** and **consider** $\mapsto$ **CONSIDER**, **apply** $\mapsto$ **APPLY**, **done** $\mapsto$ **END** or **NEXT**, **have** $\mapsto$ **HAVE**, **proof** $\mapsto$ **INTRO** ...
4) Refinement. Tactic applications still remain after step 3. To obtain more pure declarative proofs, we eliminate these tactics by repeatedly applying the following substitution until further substitution would cause **END** to fail to prove the goal using the ATP.

$$\begin{aligned} &\textbf{APPLY}\ \textit{tactic}\ \textbf{END WITH}\ w\ \textbf{WITHOUT}\ wo \\ \mapsto\ &\textbf{END WITH}\ w\ \textit{lem}^{+}\ \textbf{WITHOUT}\ wo\ \textit{lem}^{-} \end{aligned} \tag{3}$$

where lem^{+}, lem^{-} are respectively positive and negative lemma arguments of $tactic$, indicating lemmas to use and avoid, e.g., `auto add:` lem^{+} `del:` lem^{-}. Additionally, some common tactics have specialized substitutions mapping them into specific MiniLang statements, e.g.,

$$\begin{aligned} &\textbf{APPLY}\ (\texttt{auto}\ \textit{args}) \mapsto \textbf{SIMPLIFY}\ \textit{args} \\ &\textbf{APPLY}\ (\texttt{simp}\ \textit{args}) \mapsto \textbf{SIMPLIFY}\ \textit{args} \\ &\textbf{APPLY}\ (\texttt{unfold\_tac}\ \textit{defs}) \mapsto \textbf{UNFOLD}\ \textit{defs} \end{aligned}$$

B. Analysis of the Translation Result

Many translation rules are incomplete or unsafe. Consequently, the resulting translations may not always pass Isabelle's proof check. Failed proofs are discarded, and we successfully obtain 85.28% of the entire AFP corpus.

To analyze the quality of the translation in depth, we examine the occurrence frequency of every command in the translated corpus, as shown in fig. 4. END and NEXT are the most frequent commands. This is because (1) every goal and subgoal must be closed by END or NEXT, and (2) 42.65% of AFP proofs are simple enough to be proven directly by Sledgehammer*, requiring only a single END statement.

HAVE constitutes the next largest portion. This confirms its central role in declarative proofs, thus providing a metric for evaluating how well our translation preserves declarative structures: the translation success rate over proofs containing different numbers of **have** and **hence** (which are the source of HAVE). Illustrated in fig. 5, when the number of **have** and **hence** increases, the translation success rate decreases rapidly. This is expected because more complex proofs introduce additional corner cases not covered by our translation

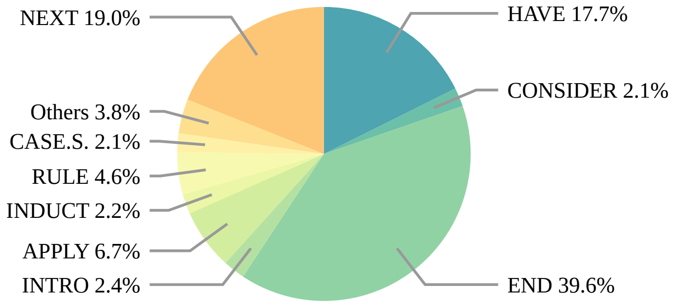


Fig. 4: Distribution of MiniLang Commands in the Translation

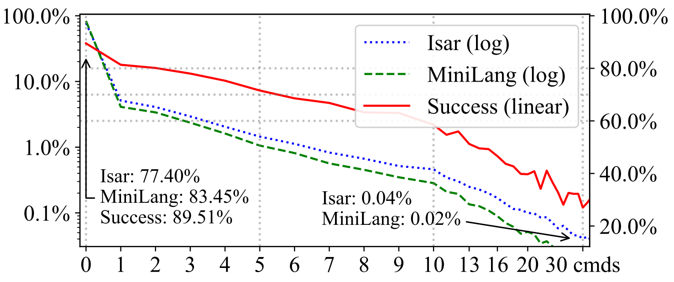


Fig. 5: Solid line: translation success rate on proofs containing different numbers of **have** and **hence** commands (cmds). Dashed and dot lines: Distributions of the number of **HAVE** and **have** + **hence** in the translated and the original proofs.

rules. Nonetheless, this number is not more than 5 for most proofs ($> 93.0\%$), where the success rate still remains above 71.5%. Thus, this translation process is still acceptable in preserving most declarative proofs seen in the corpus.

Finally, the portion of APPLY reduces from 37.2% to 6.7%. It means a large number of tactics that harm the purity of declarative proofs are eliminated, though some of them are still present and demand more translation rules to cover them.

V. Evaluation

Having obtained a substantial corpus through translation, we evaluate the impact of proof language redesign on NTP performance by building and comparing NTP systems over both Isar and MiniLang. Based on the evaluation result, we answer the following research questions:

RQ1: Can we effectively improve the declarative NTP by redesigning the underlying proof language?
RQ2: Is improvement confined to syntax error reduction or are reasoning errors also reduced?
RQ3: For a freshly crafted proof language, how can we obtain effective training corpora for this language?

A. NTP & Evaluation Setup

While numerous design choices exist for building NTP systems [4], [39], [44], best practices for real-world proof engineering contexts remain unclear. To minimize uncertainties in this evaluation, we adopt a methodology for LLM prompting and context provision inspired by Baldur [2], a leading prior work in LLM-based NTP on Isabelle (§V-A4). We conduct

Supervised Fine-Tuning (the most basic approach in LLM-based NTPs) to perform whole proof generation in a setting analogous to Baldur. However, we exclude Baldur's proof repair method to focus purely on the language comparison.

1) Base Models: To assess MiniLang over multiple models, we fine-tune two 7B base models: Llemma [45] and DeepSeek-Prover-Base v1.5 (DPSK-PB) [39]. Both models specialize in formal reasoning, having been pre-trained on datasets from major proof assistants including Coq, Lean, and Isabelle.

2) Datasets: We use Isabelle AFP version 2025-02-12 [36], Isabelle 2024 HOL libraries [35], and their MiniLang translations as our fine-tuning datasets. Unreachable code, proofs of the benchmark targets such as PISA, and long proofs that exceed the LLMs' context window are removed. Our Isar fine-tuning dataset based on the AFP is made up of $\sim$332K proofs. During the ablation study detailed in §V-B, we convert this dataset into separate versions for each proof language detailed in table II, resulting in four datasets in total, so that each experiment is conducted with a model fine-tuned to that particular proof language.

We use PISA [11] as our evaluation benchmark. PISA originally contains 3K goals randomly selected from Isabelle AFP version 2022-12-06. Due to ongoing development of Isabelle and AFP, some goals have been moved or removed from the newer version. We manually updated the dataset to AFP version 2025-02-12, removing goals that are no longer available, resulting in 2,962 goals.

3) Data Contamination: After private conversations with authors of DPSK-PB, we believe it is likely that PISA is present in their training data, as no deliberate steps were taken to exclude it. Llemma claims explicitly that they removed from their training data any proofs whose names appear in PISA [11].

Nevertheless, any data contamination is more likely to falsely inflate the performance of Isar compared to MiniLang, since MiniLang programs do not appear in either DPSL-PB or Llemma's training data, and we have ensured that PISA is not present in the fine-tuning MiniLang corpus. Thus, such contamination would minimally undermine the conclusion of the paper if the performance of MiniLang is significantly better than that of Isar.

4) Prompt Setup: Following Baldur's approach, we use a simple prompt setup with two parts: context and goal. Context includes declarations, lemmas, and proofs immediately preceding the goal in the same file. The goal contains the name and statement. Given the 4K token context window for both Llemma and DPSK-PB, we reserve 2K tokens each for context and goal, truncating distant content when necessary.

5) Supervised Fine-Tuning (SFT): We use LLaMA-Factory [46], a widely used LLM training framework to train Llemma and DPSK-PB with supervised fine-tuning. Both models are fine-tuned for 2 epochs with a batch size of 256. The learning rate is set to 2×10^{-5} , and linearly scaled to 0 during training. The training is run on 8 Nvidia H200 GPUs, and each takes around 12 hours to finish.

TABLE II: PISA evaluation of models over MiniLang and Isar

Base Model	Lang.	pass@1	pass@8
DPSK-PB	MiniLang	**69.1%**	**79.2%**
DPSK-PB	Isar + SH*	63.9%	74.3%
DPSK-PB	MiniLang - SH*	35.5%	44.9%
DPSK-PB	Isar	40.2%	50.5%
Llemma	MiniLang	**68.0%**	**78.9%**
Llemma	Isar + SH*	63.3%	72.1%
Llemma	MiniLang - SH*	35.2%	44.6%
Llemma	Isar	38.6%	48.6%

6) Proof Check: We sample a model's inference k times for each benchmark entry to obtain the pass@k. We run Isabelle to check every sample. If any of the samples passes the proof check, the entry is considered passed. The pass@k is then the portion of the passed entries in the PISA test set.

Our extended Sledgehammer* includes a self-learning system that maintains local databases of premise-goal connections. Performance on a goal improves when Sledgehammer* has previously encountered similar problems. For fair comparison, we reset Sledgehammer*'s database before each model evaluation. Nonetheless, this also means our results underestimate real-world performance, where Sledgehammer* would retain contextual knowledge and perform better.

B. Result

Based on the evaluation results of the SFT over Isar and MiniLang as listed in table II, we answer the three research questions posed at the beginning.

1) RQ1. Effectiveness of Redesigning Proof Language: To answer RQ1, we compare our redesigned language MiniLang against Isabelle's original language Isar. The results demonstrate substantial improvements of $\sim$29% across both base models and evaluation metrics.

Furthermore, to better understand the sources of this improvement, we conduct an ablation study. Prior work [29], [31] has demonstrated that the adoption of ATP tools like Sledgehammer can bring significant performance gains. Since MiniLang incorporates Sledgehammer* by design while Isar does not, we create two sets of comparisons: In the first set, we apply the same substitution principle as formula (3) to create Isar + SH*, which replaces tactics in Isar proofs with Sledgehammer* while preserving other declarative structures. This enables comparison between MiniLang and a hypothetical version of Isar with equivalent ATP support. In the second set, we disable substitution (3) and any Sledgehammer* calls to obtain MiniLang - SH*, allowing comparison with Isar without ATP enhancement. We emphasise that MiniLang is designed from the ground up to be used with ATP support, so these numbers are only presented to help separate out the sources of MiniLang's performance improvements.

When Sledgehammer* is equally enabled, corresponding to a declarative NTP scenario, MiniLang constantly outperforms

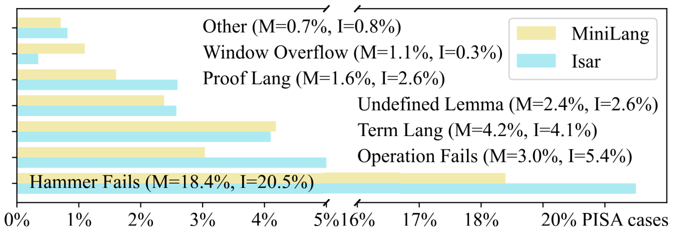


Fig. 6: Failures in the fine-tuned DPSK-PB and Llemma.

Isar across both base models and evaluation metrics (pass@1 and pass@8) by ∼5%. This is sufficient to show:

RA1: Redesigning the proof language can effectively improve the performance of declarative NTP.

When Sledgehammer* is disabled, proofs rely completely on tactic applications, turning the scenario into tactic-based NTP. In this case, MiniLang no longer maintains its advantage and falls behind Isar by ∼5%. This is reasonable given that MiniLang is specifically designed for declarative proofs.

Particularly, MiniLang restricts every tactic application to act only on the leading subgoal. While this brings clearer proof structure and better alignment between tactics and subgoals, it also causes difficulties for proof generation. LLMs can no longer use one terminating tactic (e.g., `fastforce`, `auto`) to conclude all subgoals. Instead, LLMs must repeat the tactic exactly n times for n subgoals. LLMs must precisely simulate the effect of every tactic application to trace the proof state, to know how many subgoals remain. This is extremely difficult without interaction with proof assistants, even for humans. Indeed, 37.4% proof failures are due to premature proof termination with unproven subgoals remaining — models incorrectly assume that the generated tactics are sufficient when they are not. In contrast, in Isar, LLMs can always mindlessly apply terminating tactics to try to conclude all subgoals, regardless of the number of subgoals and the shape of the proof state. Therefore, the performance loss of MiniLang - SH* is reasonable, since MiniLang's elaboration properly exposes the inherent difficulties of tactic-based NTP.

Finding: A beneficial proof language for declarative NTP is not necessarily beneficial for tactic-based NTP.

2) RQ2. Sources of Performance Improvement: We compare the failure causes of proofs generated by MiniLang and Isar + SH* in the pass@1 test. The result is shown in fig. 6. MiniLang significantly reduces syntax errors in the proofs. However, this accounts for only 1% of the 5% total improvement. More significantly, MiniLang reduces failures in proof operations and Sledgehammer* calls, meaning the generated MiniLang proofs are more likely to be correct. This indicates that the language design helps LLMs generate higher-quality proofs, not merely syntactically correct text.

RA2: The improvement brought by redesigning the proof language can extend beyond syntax errors to enhance models' capability of generating logically correct proofs

We also observe a slight increase in term language errors for MiniLang, despite no modifications to this aspect of the language. This likely reflects the sequential nature of our error analysis, which reports only the first error encountered — as MiniLang resolves certain types of errors, other unrelated issues may become more visible in the error categorization.

3) RQ3: Training Corpora for a New Proof Language: The MiniLang training corpus is obtained by translating Isabelle's AFP. The translation process is incomplete, losing ∼15% of proofs, with higher loss rates observed for longer declarative proofs (fig. 4). Despite this data reduction, MiniLang's experimental results still demonstrate clear advantages due to language improvements. This validates the effectiveness of our translation-based approach.

RA3: Rule-based translation is an effective approach to obtain training corpora for freshly crafted proof languages.

VI. Related Works

A. NTP for Math Competitions vs. Real-world Engineering

Early NTP works stem from the intersection of the time-honored interactive theorem proving and the emerging machine learning field, mainly pushed by researchers who wish to improve the automation of the provers [43], [47]–[53].

Later, when LLMs achieved breakthrough progress in numerous domains including math, researchers took notice of models' formal reasoning capabilities and introduced a series of competition-based benchmarks [5], [7], [8]. In the following years, NTP works focusing on math competitions emerged in abundance [3], [4], [39]–[41], [44], [54].

However, math competitions cannot represent real-world proof engineering. Substantial gaps exist between the two. First, math competitions consider only a fixed and relatively small domain of textbook-level concepts in number theory, algebra, analysis, etc. By contrast, real proof engineering confronts unlimited and diverse cutting-edge concepts across math, programming languages, computer security, etc. Second, it remains doubtful whether competition problem-solving skills translate to real-world proof engineering capabilities. Researchers [55]–[58] have reported that models' impressive performance on informal math benchmarks may stem from rote memorization of question patterns rather than genuine mathematical understanding. It is unclear whether models similarly rely on memorizing proof tactics tailored to specific competition problems rather than acquiring the authentic skills that proof engineering requires.

B. Analysis to LLMs' challenges in learning proof languages

PALM [1] conducts a formative study about failures of a GPT-based NTP over Coq's proof language Gallina. First, this study is based on analysing the direct reasons causing the failures. It remains uncertain whether these failures stem from more fundamental issues, such as conflicts between the proof assistant's working model and the LLM's inherent limitations in symbolic reasoning. By contrast, our work examines Isabelle's proof language Isar design issues through a combined theoretical and experimental approach, offering deeper insights while acknowledging that a fundamental understanding remains elusive. Second, PALM's findings focus on Gallina, a tactic-based language, while our work focuses on declarative proofs. Some issues found by PALM (e.g., bullet and theorem application) cannot be directly applied to declarative Isar because some notions do not exist in Isar at all. For example, Isar has no bullet; the theorem application, which represents 49.6% of failure reasons in PALM, is not a primary element in declarative proofs.

C. *Proof Languages & Representations for Machine Learning*

Many works [4], [17], [50], [59], [60] use their own representations mixed with proof scripts, proof states, and/or human solvers' informal thoughts. Some works transform the syntactical representations of the language, like Passport's tree representation of Coq [61], graph representation of HOL [62], and the S-expressions adopted by CoqGym [12], MLFMF [63], and HOList [64]. However, they do not change the core (semantics, proof model) of their proof languages.

Regarding redesigning a proof language to improve NTP's performance, the paper is the first to the best of our knowledge.

D. *Language Models for Real-World Proof Generation*

Pioneers in applying LLMs to generate proofs include Urban *et al.* [51] and Polu *et al.* [52]. Later, a series of benchmarks sourced from real-world proof engineering are proposed, such as PISA [11] for Isabelle/HOL, LeanDojo [19] for Lean, and CoqGym [12] for Coq. Targeting these benchmarks, abundant works have emerged.

Leading works on PISA and Isabelle/HOL derive from the declarative NTP approach established by Thor [29]. Thor exhaustively replaces tactics and other language components with Sledgehammer to obtain declarative proofs, which are then used to train LLMs. Thor reaches a success rate of 57.0% on PISA. Baldur [2] extends Thor's approach by fine-tuning the LLM Minerva to generate whole Isabelle/HOL proofs. It also incorporates a repair model that leverages Isabelle's error messages to fix broken proofs. Combining Thor's approach, Baldur reaches 65.7% with pass@64 on PISA. Magnushammer [65] adopts contrastive learning to target premise selection, the same task as Sledgehammer. Though the topic itself is irrelevant to the proof generation by LLMs, its combination with Thor reaches the previous state-of-the-art, a success rate of 71% on PISA. Our work applies supervised fine-tuning in a similar setting to Baldur, with differences: 1) we use MiniLang as the proof language; 2) unlike Thor, MiniLang eliminates only tactics while preserving other declarative statements; 3) we employ more recent base models; 4) we do not incorporate proof repair mechanisms. Under a smaller computation budget (500s timeout, 16 CPU cores, and up to 8 attempts) than Magnushammer, we achieve a success rate of 79.0%. Our end-to-end results represent new state-of-the-art for PISA performance.

Works in Coq, Lean, and Metamath either follow a search-based approach or generate whole proofs including all tactics.

Most search-based works stem from Polu and Sutskever's stepwise best-first proof search [52]: at each iteration, they select and expand the open subgoal whose sequence of generated proof-steps has the highest cumulative log-probability under the model. It reaches 56.5% with pass@32 on 1000 randomly selected Metamath goals from set.mm library. ReProver [19] improves the search by introducing premise selection at every iteration. It reaches 51.2% with pass@1 on LeanDojo's random split and 26.3% on the novel-premise split. Lean-STaR [66] integrates informal thoughts and reaches 39.4% with pass@1 on LeanDojo's novel-premise.

Regarding whole proof generation, PAML [1] follows a generate-then-repair approach similar to Baldur, but uses a rule-based repair mechanism instead of another LLM.

VII. Discussion

A. *Potentials in Extending MiniLang to Other Proof Assistants*

While this work implements MiniLang in Isabelle/HOL, it is possible to port at least the core of MiniLang to other proof assistants like Lean. The key is that MiniLang targets declarative proof, where the notions are general across specific logics and software systems. Although some commands (e.g., **CONFIG**) are specifically designed for compatibility with Isar proofs, most MiniLang commands have their correspondences in other proof assistants. Examples include Lean's `have`, `simp`, `use`, `intros`, `cases`, and `induction`. Relying on powerful ATPs, we believe most disparities between proof assistants can be harmonized. Hopefully, MiniLang can serve as a bridge, preventing the long-standing fragmentation in the field of proof assistants from spreading into the field of NTP.

VIII. Limitations and Future Work

A benefit of MiniLang's proof model is that its proof state integrates all subgoals and contextual information in a structural way. This paper focused on whole proof generation. Future work could explore Baldur-style proof repair, or stepwise approaches that could leverage this proof state structure, such as reinforcement learning.

In the whole proof generation setting, further work could be done to refine the provided LLM context - the current approach simply takes the most recent items from the same file. Better premise selection could look across multiple files to create contexts with more relevant information.

We leave these mentioned aspects to our future work.

Acknowledgment

The REPL infrastructure, MiniLang interpreter, translator, training data, machine learning framework, Sledgehammer*, and parameters of the finetuned models will be open-sourced and are present in the submitted supplementary materials.

References

[1] M. Lu, B. Delaware, and T. Zhang, "Proof automation with large language models," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1509–1520.

[2] E. First, M. N. Rabe, T. Ringer, and Y. Brun, "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*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 1229–1241.

[3] Z. Z. Ren, Z. Shao, J. Song, H. Xin, H. Wang, W. Zhao, L. Zhang, Z. Fu, Q. Zhu, D. Yang, Z. F. Wu, Z. Gou, S. Ma, H. Tang, Y. Liu, W. Gao, D. Guo, and C. Ruan, "DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition," 2025. [Online]. Available: https://arxiv.org/abs/2504.21801

[4] H. Wang, M. Unsal, X. Lin, M. Baksys, J. Liu, M. D. Santos, F. Sung, M. Vinyes, Z. Ying, Z. Zhu, J. Lu, H. de Saxcé, B. Bailey, C. Song, C. Xiao, D. Zhang, E. Zhang, F. Pu, H. Zhu, J. Liu, J. Bayer, J. Michel, L. Yu, L. Dreyfus-Schmidt, L. Tunstall, L. Pagani, M. Machado, P. Bourigault, R. Wang, S. Polu, T. Barroyer, W.-D. Li, Y. Niu, Y. Fleureau, Y. Hu, Z. Yu, Z. Wang, Z. Yang, Z. Liu, and J. Li, "Kimina-prover preview: Towards large formal reasoning models with reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2504.11354

[5] K. Zheng, J. M. Han, and S. Polu, "miniF2F: a cross-system benchmark for formal Olympiad-level mathematics," in *10th International Conference on Learning Representations (ICLR)*, Virtual Event, April 2022.

[6] G. Tsoukalas, J. Lee, J. Jennings, J. Xin, M. Ding, M. Jennings, A. Thakur, and S. Chaudhuri, "PutnamBench: Evaluating Neural Theorem-Provers on the Putnam Mathematical Competition," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, BC, Canada, December 2024.

[7] C. Liu, J. Shen, H. Xin, Z. Liu, Y. Yuan, H. Wang, W. Ju, C. Zheng, Y. Yin, L. Li, M. Zhang, and Q. Liu, "FIMO: A Challenge Formal Dataset for Automated Theorem Proving," 2023. [Online]. Available: https://arxiv.org/abs/2309.04295

[8] Z. Azerbayev, B. Piotrowski, H. Schoelkopf, E. W. Ayers, D. Radev, and J. Avigad, "ProofNet: Autoformalizing and Formally Proving Undergraduate-Level Mathematics," 2023. [Online]. Available: https://arxiv.org/abs/2302.12433

[9] J. Liu, X. Lin, J. Bayer, Y. Dillies, W. Jiang, X. Liang, R. Soletskyi, H. Wang, Y. Xie, B. Xiong, Z. Yang, J. Zhang, L. Zhi, J. Li, and Z. Liu, "CombiBench: Benchmarking LLM Capability for Combinatorial Mathematics," 2025. [Online]. Available: https://arxiv.org/abs/2505.03171

[10] H. Ying, Z. Wu, Y. Geng, J. Wang, D. Lin, and K. Chen, "Lean Workbook: A large-scale Lean problem set formalized from natural language math problems," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, BC, Canada, December 2024.

[11] A. Q. Jiang, W. Li, J. M. Han, and Y. Wu, "LISA: Language models of ISAbelle proofs," in *6th Conference on Artificial Intelligence and Theorem Proving (AITP)*, 2021.

[12] K. Yang and J. Deng, "Learning to Prove Theorems via Interacting with Proof Assistants," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97. PMLR, 09–15 Jun 2019, pp. 6984–6994.

[13] L. Zhang, S. Lu, and N. Duan, "Selene: Pioneering Automated Proof in Software Verification," in *ACL 2024, Bangkok, Thailand, August 11-16*, L. Ku, A. Martins, and V. Srikumar, Eds. Association for Computational Linguistics, 2024, pp. 1776–1789.

[14] E. Lohn and S. Welleck, "miniCodeProps: a Minimal Benchmark for Proving Code Properties," 2024. [Online]. Available: https://arxiv.org/abs/2406.11915

[15] J. Hu, T. Zhu, and S. Welleck, "miniCTX: Neural Theorem Proving with (Long-)Contexts," in *ICLR 2025, Singapore, April 24-28, 2025*, 2025.

[16] A. Sanchez-Stern, Y. Alhessi, L. Saul, and S. Lerner, "Generating correctness proofs with neural networks," in *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, ser. MAPL 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–10.

[17] W. Li, L. Yu, Y. Wu, and L. C. Paulson, "IsarStep: a Benchmark for High-level Mathematical Reasoning," in *ICLR 2021, Virtual Event, Austria, May 3-7*, 2021.

[18] X. Lin, Q. Cao, Y. Huang, H. Wang, J. Lu, Z. Liu, L. Song, and X. Liang, "FVEL: interactive formal verification environment with large language models via theorem proving," in *NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024.

[19] K. Yang, A. M. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. J. Prenger, and A. Anandkumar, "LeanDojo: Theorem Proving with Retrieval-Augmented Language Models," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., New Orleans, LA, USA, December 2023.

[20] S. Chakraborty, G. Ebner, S. Bhat, S. Fakhoury, S. Fatima, S. Lahiri, and N. Swamy, "Towards Neural Synthesis for SMT-Assisted Proof-Oriented Programming," 2024. [Online]. Available: https://arxiv.org/abs/2405.01787

[21] Statistics of 46 NTP-related works in arXiv since 2024, Supplementary Material. See the supplementary materials attached to this article. [Online]. Available: https://zenodo.org/records/15552130

[22] L. C. Paulson, *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*, ser. Lecture Notes in Computer Science. Springer, 1994, vol. 828.

[23] J. C. Blanchette, S. Böhme, and L. C. Paulson, "Extending sledgehammer with smt solvers," *Journal of Automated Reasoning*, vol. 51, no. 1, pp. 109–128, Jun 2013.

[24] A. Mohamed, T. Mascarenhas, H. Khan, H. Barbosa, A. Reynolds, Y. Qian, C. Tinelli, and C. Barrett, "Lean-smt: An smt tactic for discharging proof goals in lean," May 2025. [Online]. Available: https://arxiv.org/abs/2505.15796

[25] Y. Qian, J. Clune, C. Barrett, and J. Avigad, "Lean-auto: An Interface between Lean 4 and Automated Theorem Provers," May 2025. [Online]. Available: https://arxiv.org/abs/2505.14929

[26] A. Geesing, "Premise Selection for Lean 4," Master's thesis, Universiteit van Amsterdam, June 2023.

[27] K. Palmskog, "qarith-stern-brocot," https://github.com/rocq-community/qarith-stern-brocot/blob/9400dafdc7e35b53d23ab336e3a1b548c3b09133/theories/sqrt2.v, 2022, [Online; accessed 07-May-2025].

[28] Lean community, "Logic and Proof," https://leanprover-community.github.io/logic_and_proof/introduction.html, 2025, [Online; accessed 07-May-2025].

[29] A. Q. Jiang, W. Li, S. Tworkowski, K. Czechowski, T. Odrzygóźdź, P. Milos, Y. Wu, and M. Jamnik, "Thor: Wielding Hammers to Integrate Language Models and Automated Theorem Provers," in *NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[30] Wikipedia, "Isabelle (proof assistant)," http://en.wikipedia.org/w/index.php?title=Isabelle%20(proof%20assistant), 2025, [Online; accessed 07-May-2025].

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

[32] M. Parmar, N. Patel, N. Varshney, M. Nakamura, M. Luo, S. Mashetty, A. Mitra, and C. Baral, "LogicBench: Towards Systematic Evaluation of Logical Reasoning Ability of Large Language Models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, L. Ku, A. Martins, and V. Srikumar, Eds. Association for Computational Linguistics, 2024, pp. 13 679–13 707.

[33] B. Jiang, Y. Xie, Z. Hao, X. Wang, T. Mallick, W. Su, C. J. Taylor, and D. Roth, "A Peek into Token Bias: Large Language Models Are Not Yet Genuine Reasoners," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, Y. Al-Onaizan, M. Bansal, and Y. Chen, Eds. Association for Computational Linguistics, 2024, pp. 4722–4756. [Online]. Available: https://aclanthology.org/2024.emnlp-main.272

[34] N. Patel, M. Kulkarni, M. Parmar, A. Budhiraja, M. Nakamura, N. Varshney, and C. Baral, "Multi-LogiEval: Towards Evaluating Multi-Step Logical Reasoning Ability of Large Language Models," in *Proceedings of the 2024 Conference on Empirical Methods in*

Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024, Y. Al-Onaizan, M. Bansal, and Y. Chen, Eds. Association for Computational Linguistics, 2024, pp. 20 856–20 879. [Online]. Available: https://aclanthology.org/2024.emnlp-main.1160

[35] Isabelle contributors, "Hol libraries," the `src/HOL` folder in the redistribution pack of Isabelle 2024, https://isabelle.in.tum.de/website-Isabelle2024, 2024, [Online; accessed 07-May-2025].

[36] AFP contributors, "Archive of Formal Proofs," https://www.isa-afp.org/, 2025.

[37] Lean Prover Community, "mathlib4: The math library for Lean 4," https://github.com/leanprover-community/mathlib4, 2025, commit 05f6300492a69d93dcbc3e05d465c58ced9dc277 on 2025-05-27.

[38] K. Thompson, N. Saavedra, P. Carrott, K. Fisher, A. Sanchez-Stern, Y. Brun, J. F. Ferreira, S. Lerner, and E. First, "Rango: Adaptive Retrieval-Augmented Proving for Automated Software Verification," in *47th International Conference on Software Engineering (ICSE)*, Ottowa, ON, Canada, April 2025.

[39] H. Xin, Z. Z. Ren, J. Song, Z. Shao, W. Zhao, H. Wang, B. Liu, L. Zhang, X. Lu, Q. Du, W. Gao, H. Zhang, Q. Zhu, D. Yang, Z. Gou, Z. F. Wu, F. Luo, and C. Ruan, "DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search," in *ICLR 2025, Singapore, April 24-28*, 2025.

[40] H. Xin, D. Guo, Z. Shao, Z. Ren, Q. Zhu, B. Liu, C. Ruan, W. Li, and X. Liang, "Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data," 2024. [Online]. Available: https://arxiv.org/abs/2405.14333

[41] X. Zhao, L. Zheng, H. Bo, C. Hu, U. Thakker, and L. Kong, "SubgoalXL: Subgoal-based Expert Learning for Theorem Proving," 2024. [Online]. Available: https://arxiv.org/abs/2408.11172

[42] J. Meng and L. C. Paulson, "Translating higher-order clauses to first-order clauses," *Journal of Automated Reasoning*, vol. 40, no. 1, pp. 35–60, Jan 2008.

[43] D. Kühlwein, J. C. Blanchette, C. Kaliszyk, and J. Urban, "MaSh: Machine Learning for Sledgehammer," in *Interactive Theorem Proving*, S. Blazy, C. Paulin-Mohring, and D. Pichardie, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 35–50.

[44] H. Wang, H. Xin, C. Zheng, Z. Liu, Q. Cao, Y. Huang, J. Xiong, H. Shi, E. Xie, J. Yin, Z. Li, and X. Liang, "LEGO-Prover: Neural Theorem Proving with Growing Libraries," in *12th International Conference on Learning Representations, (ICLR)*, Vienna, Austria, May 2024.

[45] Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. M. McAleer, A. Q. Jiang, J. Deng, S. Biderman, and S. Welleck, "Llemma: An open language model for mathematics," in *12th International Conference on Learning Representations (ICLR)*, Vienna, Austria, May 2024.

[46] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, and Z. Luo, "LlamaFactory: Unified efficient fine-tuning of 100+ language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Y. Cao, Y. Feng, and D. Xiong, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 400–410. [Online]. Available: https://aclanthology.org/2024.acl-demos.38/

[47] C. Kaliszyk and J. Urban, "MizAR 40 for Mizar 40," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 245–256, Oct 2015.

[48] G. Irving, C. Szegedy, A. A. Alemi, N. Een, F. Chollet, and J. Urban, "DeepMath - Deep Sequence Models for Premise Selection," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, 2016.

[49] M. Färber, C. Kaliszyk, and J. Urban, "Monte Carlo Tableau Proof Search," in *Automated Deduction – CADE 26*, L. de Moura, Ed. Cham: Springer International Publishing, 2017, pp. 563–579.

[50] C. Kaliszyk, F. Chollet, and C. Szegedy, "HolStep: A Machine Learning Dataset for Higher-order Logic Theorem Proving," in *ICLR 2017, Toulon, France, April 24-26, 2017*, 2017.

[51] J. Urban and J. Jakubův, "First neural conjecturing datasets and experiments," in *Intelligent Computer Mathematics*, C. Benzmüller and B. Miller, Eds. Cham: Springer International Publishing, 2020, pp. 315–323.

[52] S. Polu and I. Sutskever, "Generative language modeling for automated theorem proving," 2020. [Online]. Available: https://arxiv.org/abs/2009.03393

[53] Z. A. Goertzel, J. Jakubův, C. Kaliszyk, M. Olšák, J. Piepenbrock, and J. Urban, "The Isabelle ENIGMA," in *13th International Conference on Interactive Theorem Proving (ITP 2022)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. Andronick and L. de Moura, Eds., vol. 237. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 16:1–16:21.

[54] B. Rao, W. Eiers, and C. Lipizzi, "Neural Theorem Proving: Generating and Structuring Proofs for Formal Verification," 2025. [Online]. Available: https://arxiv.org/abs/2504.17017

[55] S. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar, "GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models," in *ICLR 2025, Singapore, April 24-28*, 2025.

[56] K. Huang, J. Guo, Z. Li, X. Ji, J. Ge, W. Li, Y. Guo, T. Cai, H. Yuan, R. Wang, Y. Wu, M. Yin, S. Tang, Y. Huang, C. Jin, X. Chen, C. Zhang, and M. Wang, "Math-perturb: Benchmarking llms' math reasoning abilities against hard perturbations," 2025. [Online]. Available: https://arxiv.org/abs/2502.06453

[57] E. S. Salido, J. Gonzalo, and G. Marco, "None of the others: a general technique to distinguish reasoning from memorization in multiple-choice llm evaluation benchmarks," 2025. [Online]. Available: https://arxiv.org/abs/2502.12896

[58] C. Xie, Y. Huang, C. Zhang, D. Yu, X. Chen, B. Y. Lin, B. Li, B. Ghazi, and R. Kumar, "On memorization of large language models in logical reasoning," 2025. [Online]. Available: https://arxiv.org/abs/2410.23123

[59] T. Gauthier, C. Kaliszyk, J. Urban, R. Kumar, and M. Norrish, "Tactictoe: Learning to prove with tactics," *Journal of Automated Reasoning*, vol. 65, no. 2, pp. 257–286, Feb 2021.

[60] S. Welleck and R. Saha, "Llmstep: Llm proofstep suggestions in lean," 2023. [Online]. Available: https://arxiv.org/abs/2310.18457

[61] A. Sanchez-Stern, E. First, T. Zhou, Z. Kaufman, Y. Brun, and T. Ringer, "Passport: Improving Automated Formal Verification Using Identifiers," *ACM Trans. Program. Lang. Syst.*, vol. 45, no. 2, Jun. 2023.

[62] A. Paliwal, S. Loos, M. Rabe, K. Bansal, and C. Szegedy, "Graph Representations for Higher-Order Logic and Theorem Proving," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, pp. 2967–2974, Apr. 2020.

[63] A. Bauer, M. Petkovic, and L. Todorovski, "MLFMF: Data Sets for Machine Learning for Mathematical Formalization," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023.

[64] K. Bansal, S. Loos, M. Rabe, C. Szegedy, and S. Wilcox, "HOList: An Environment for Machine Learning of Higher Order Logic Theorem Proving," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 454–463. [Online]. Available: https://proceedings.mlr.press/v97/bansal19a.html

[65] M. Mikula, S. Tworkowski, S. Antoniak, B. Piotrowski, A. Q. Jiang, J. P. Zhou, C. Szegedy, L. Kucinski, P. Milos, and Y. Wu, "Magnushammer: A Transformer-Based Approach to Premise Selection," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[66] H. Lin, Z. Sun, S. Welleck, and Y. Yang, "Lean-star: Learning to interleave thinking and proving," in *ICLR 2025, Singapore, April 24-28*, 2025.