What are the Right Symmetries for Formal Theorem Proving?

Krzysztof Olejniczak [1] **Radoslav Dimitrov** **Xingyue Huang** [1]
Bernardo Cuenca Grau [1] **Jinwoo Kim** [2] **İsmail İlkan Ceylan** [3 4 1]

Abstract

Formal theorem provers based on large language models (LLMs) are highly sensitive to superficial variations in problem representation: semantically equivalent statements can exhibit drastically different proof success rates, revealing a failure to respect structural symmetries inherent in formal mathematics. This raises a central question: *what are the right symmetries for formal theorem proving?* We introduce *rewriting categories*, a category-theoretic framework capturing the compositional, generally non-invertible transformations induced by proof tactics, and use it to formalize two symmetry notions: *proof equivariance*, governing how proof distributions transform under rewrites, and *success invariance* (i.e., invariance of success probability), requiring equivalent statements to be solved with the same probability. We observe that state-based next-tactic provers naturally satisfy proof equivariance by operating on proof states. In contrast, state-of-the-art LLM-based provers satisfy neither property, exhibiting large performance variation across equivalent formulations. To mitigate this, we propose test-time methods that aggregate over equivalent rewritings of the input, showing theoretically that they recover success invariance in the sampling limit, and empirically, that they improve robustness and performance under fixed inference budgets. Our results highlight symmetry as a key missing inductive bias in LLM-based theorem proving and suggest test-time computation as a practical route to approximate it.

[1]Department of Computer Science, University of Oxford [2]KAIST [3]TU Wien [4]AITHYRA. Correspondence to: Krzysztof Olejniczak <krzysztof.olejniczak@cs.ox.ac.uk>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

1. Introduction

Large language model (LLM) theorem provers, operating on textual representations of formal statements, are highly sensitive to how problems are expressed (Zhao et al., 2025; Tian et al., 2026). As shown in Figure 2, even simple, semantics-preserving rewrites, such as reordering arguments or applying elementary algebraic transformations, can lead to drastic changes in proof success rates. In some cases, a theorem that is solved nearly always becomes almost unsolvable under another formulation, despite the two statements being mathematically equivalent. This behavior, widely observed in mathematical reasoning and language understanding (Mirzadeh et al., 2025; Irpan et al., 2025; Berglund et al., 2024; Wu et al., 2025; Wei et al., 2025), reveals a fundamental limitation of current systems: failure to generalize across different representations of the same problem.

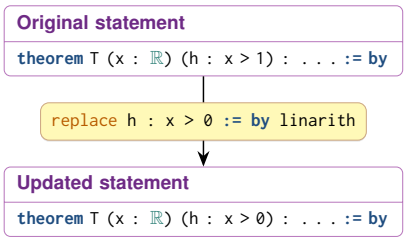

Figure 1. An example of tactic irreversibility in Lean. Applying a tactic transforms $x > 1$ into $x > 0$, losing information.

LLM-based provers (Polu and Sutskever, 2020; Yang et al., 2023; Xin et al., 2025a) process statements as text, making their behavior depend on surface form instead of the underlying mathematical structure. This discrepancy motivates a central question: *what are the right symmetries for formal theorem proving?* In classical machine learning, symmetries are typically modeled via group actions (Bronstein et al., 2021), which capture invertible transformations such as rotations or permutations. However, this framework is insufficient for formal reasoning. In proof assistants such as Lean (de Moura and Ullrich, 2021), transformations between statements arise via sequences of tactics, which are generally non-invertible and may discard information (see Figure 1). Consequently, the resulting transformation space cannot be characterized by group symmetries alone.

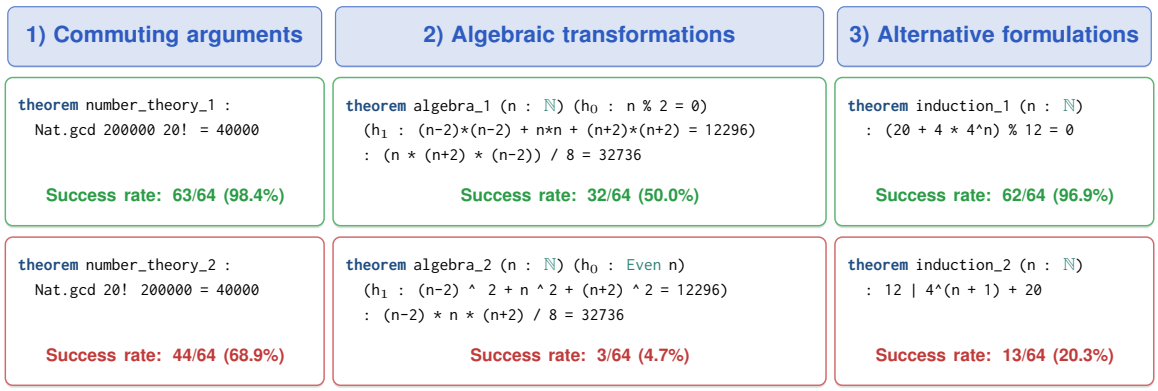

Figure 2. Sensitivity of DeepSeek-Prover-V2 (Ren et al., 2025) to semantics-preserving rewrites. Each pair of statements is mathematically equivalent, yet exhibits drastically different proof success rates under minor transformations. This highlights a systematic violation of symmetry: the prover's success probability depends strongly on the representation instead of the underlying mathematical content.

In this work, we introduce *rewriting categories*, a category-theoretic framework for modeling transformations between theorem statements. This perspective allows us to formalize two symmetry notions for formal theorem provers: **proof equivariance**, which requires proof distributions to transform consistently along rewrites, and **success invariance** (i.e., invariance of success probability), which requires equivalent statements to be solved with the same probability. Proof equivariance is naturally satisfied by next-tactic predictors (Yang et al., 2023; Li et al., 2024; Xin et al., 2025b) when used generate proofs sequentially from intermediate proof states. On the other hand, modern state-of-the-art LLM-based provers (Ren et al., 2025; Lin et al., 2025; 2026; Wang et al., 2025) satisfy neither property, exhibiting large variability across equivalent formulations.

We show that this lack of symmetry can be mitigated by aggregating over equivalent rewritings of the input statement. We prove that such procedures recover success invariance in the sampling limit, and empirically demonstrate improved robustness and performance under fixed inference budgets. These results suggest that symmetry, defined in a categorical sense, is a key missing inductive bias in current LLM-based theorem provers, and that it can be effectively approximated through test-time computation without modifying the underlying model. Our contributions can be summarized as:

(1) **Category-theoretic framework.** We introduce rewriting categories, a category-theoretic framework for modeling transformations between formal theorem statements, and formalize two symmetry notions: proof equivariance and success invariance.
(2) **Empirical analysis of invariance.** We construct miniF2F-rw, a benchmark consisting of semantically equivalent reformulations of formal problems, and show that state-of-the-art LLM-based provers exhibit large violations of success invariance.
(3) **Test-time invariance.** We propose a simple, model-agnostic test-time procedure that aggregates over equivalent rewritings of the input, and theoretically prove that it recovers success invariance in the sampling limit.
(4) **Competitive performance.** We demonstrate empirically that the proposed method improves both robustness to input reformulations and overall proof success under fixed inference budgets.

2. Formal theorem proving in Lean

Theorem proving in Lean. Formal theorem proving in Lean is structured around transformations of proof states (de Moura and Ullrich, 2021). A theorem statement is compiled into an initial state representing assumptions and goals, and proofs are constructed by applying sequences of tactics that transform this state until no goals remain. This operational view exposes a key structural property: transformations between statements arise through sequences of tactics, which are compositional but generally non-invertible.

Formally, let Stmt denote the set of all valid theorem statements, $\mathcal{T}$ the set of all tactics, and Ω the set of all Lean states, with Ω containing an *empty state* $\varnothing$ (no remaining goals) and an *error state* $\perp$ (failure during compilation). Given a theorem T, we will write $\mathcal{L}(T)$ for the corresponding state.

Each tactic $t \in \mathcal{T}$ can be seen as a function $t : \Omega \to \Omega$. In this view, a sequence of tactics $\mathbf{t} = (t_1, \ldots, t_k)$ is a valid proof of a statement T, if their composition maps $\mathcal{L}(T)$ to the empty state:

$$\mathbf{t}(\mathcal{L}(T)) = (t_k \circ t_{k-1} \circ \cdots \circ t_1)(\mathcal{L}(T)) = \varnothing.$$

Formal LLM-based provers. We focus on single-pass whole-proof generation models (Ren et al., 2025; Lin et al., 2025; 2026; Wang et al., 2025). In this setting, an LLM defines a distribution $p_\theta(\cdot \mid T)$ over sequences of tactics (i.e., candidate proofs), conditioned on a Lean statement T. The success rate $s_\theta(T)$ is then the probability that a sampled

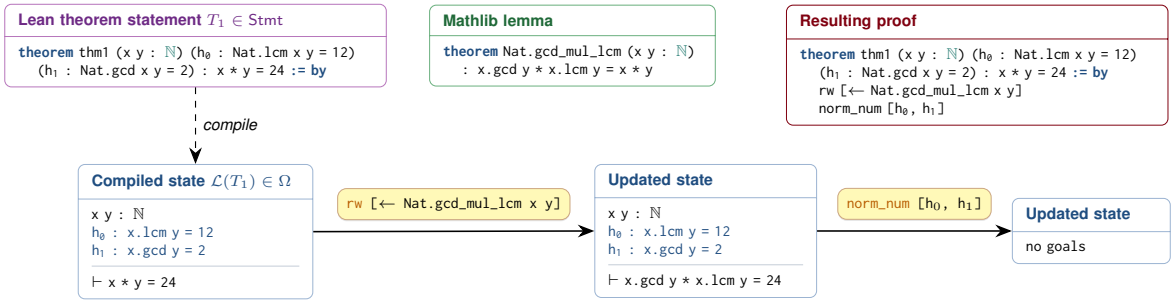

Figure 3. An example of a successful proof in Lean. The original statement T is converted into an initial state $\mathcal{L}(T)$, and tactics are then sequentially executed, until the empty state $\varnothing$ is reached.

tactic sequence $\mathbf{t}$ transforms the initial state $\mathcal{L}(T)$ into $\varnothing$:

$$s_\theta(T) \;:=\; \sum_{\mathbf{t}} p_\theta(\mathbf{t} \mid T)\, \mathbf{1}(\mathbf{t}(\mathcal{L}(T)) = \varnothing)$$

The structure of Lean statement space. In Lean, transformations between statements arise through sequences of tactics acting on proof states, linking statements T and T' whenever $\mathcal{L}(T') = \mathbf{t}(\mathcal{L}(T))$. While it is tempting to model this structure via group actions, such an abstraction is inappropriate. These transformations are compositional but generally non-invertible: multiple statements may map to the same state, and tactic application can discard information, preventing reversal. As a result, the transformation structure cannot be captured by group symmetries.

3. Formalization of mathematical symmetries via rewriting categories

We now formalize the transformations described in Section 2 using category theory. We model theorem statements as objects and transformations between them as composable, generally non-invertible morphisms. This induces a categorical structure that captures the behavior of tactics, which we refer to as a rewriting category. This framework provides the foundation for defining symmetry in formal theorem proving and explains inductive biases present in prior works as special cases.

3.1. Category theory preliminaries

We recall the basic categorical notions needed for our construction; see Appendix A for more details.

Definition 1 (Category). A *category* $\mathscr{C}$ consists of

- a collection of *objects* $\mathrm{ob}(\mathscr{C})$,
- for each $A, B \in \mathrm{ob}(\mathscr{C})$, a collection $\mathscr{C}(A, B)$ of *maps* or *arrows* from A to B,
- for each pair of arrows $f \in \mathscr{C}(A, B)$ and $g \in \mathscr{C}(B, C)$, their *composition* $g \circ f \in \mathscr{C}(A, C)$,
- for each object $A \in \mathrm{ob}(\mathscr{C})$, an element 1_A of $\mathscr{C}(A, A)$, called the *identity* on A,

satisfying the following axioms:

- *associativity*: $(h \circ g) \circ f = h \circ (g \circ f)$ for any $f \in \mathscr{C}(A, B)$, $g \in \mathscr{C}(B, C)$ and $h \in \mathscr{C}(C, D)$,
- *identity laws*: for each $f \in \mathscr{C}(A, B)$, we have $f \circ 1_A = f = 1_B \circ f$.

We often write $A \in \mathscr{C}$ to mean $A \in \mathrm{ob}(\mathscr{C})$, and $f : A \to B$ or $A \xrightarrow{f} B$ to mean $f \in \mathscr{C}(A, B)$. We will also write $A \leftrightarrows B$ for objects $A, B \in \mathscr{C}$ such that there exist arrows $A \to B$ and $B \to A$ in $\mathscr{C}$. When for all objects $A, B \in \mathscr{C}$, there is an arrow $A \to B$ if and only if there is an arrow $B \to A$, we call $\mathscr{C}$ *reciprocal*. If categories $\mathscr{C}$ and $\mathscr{D}$ satisfy $\mathrm{ob}(\mathscr{C}) \subseteq \mathrm{ob}(\mathscr{D})$, and for every $A, B \in \mathscr{C}$, we have $\mathscr{C}(A, B) \subseteq \mathscr{D}(A, B)$, we call $\mathscr{C}$ a *subcategory* of $\mathscr{D}$, denoted as $\mathscr{C} \subseteq \mathscr{D}$.

Example 2. There is a category $\mathbf{Set}$ whose objects are sets, whose maps are functions between sets, with function composition as composition and identity functions as identities.

Example 3. There is a *full tactic rewriting category* $\mathscr{R}_{\mathcal{T}}$, whose objects are Lean statements Stmt and arrows represent sequences of tactics. More precisely, if statements $T, T' \in$ Stmt satisfy $\mathcal{L}(T') = \mathbf{t}(\mathcal{L}(T))$ for some sequence of tactics $\mathbf{t}$, there is an arrow $\mathbf{t} : T \to T'$. Identities correspond to the empty sequence of tactics, representing no action. Composition becomes concatenation:

$$\mathbf{t}' \circ \mathbf{t} = [\mathbf{t} : \mathbf{t}'] \qquad \text{whenever} \qquad T \xrightarrow{\mathbf{t}} T' \xrightarrow{\mathbf{t}'} T''$$

To let categories act on concrete data, we use structure-preserving maps, called *functors*:

Definition 4 (Functor). A functor $F : \mathscr{C} \to \mathscr{D}$ consists of an object map $F : \mathrm{ob}(\mathscr{C}) \to \mathrm{ob}(\mathscr{D})$, and, for each pair $A, B \in \mathrm{ob}(\mathscr{C})$, an assignment on arrows, $F : \mathscr{C}(A, B) \to \mathscr{D}(F(A), F(B))$, satisfying:

- *preservation of composition*: $F(g \circ f) = F(g) \circ F(f)$ whenever $A \xrightarrow{f} B \xrightarrow{g} C$ in $\mathscr{C}$,
- *preservation of identities*: $F(1_A) = 1_{F(A)}$ for every $A \in \mathscr{C}$.

Example 5. There is an *implementation functor* $\mathcal{I} : \mathscr{R}_{\mathcal{T}} \to \mathbf{Set}$ mapping each theorem $T \in$ Stmt to the singleton set $\{T\}$, while arrows are mapped to the unique functions between singletons.

3.2. Rewriting categories

With the language of Section 3.1, we introduce the specific categorical objects we will use to reason about semantics-preserving rewrites of theorem statements. Our goal is to define categories $\mathscr{R}$ with Lean statements as objects, and arrows encoding transformations induced by tactics.

The maximal semantic rewriting category. Every $T \in \mathsf{Stmt}$ carries intended mathematical content, and we posit an abstract *maximal semantic rewriting category* $\mathscr{R}^*$ whose objects are statements, and arrows are all mathematically-sound transformations (including, but not limited to Lean tactics). Concretely, there is an arrow $a : T \to T'$ in $\mathscr{R}^*$ for any logically valid argument a reducing T to T'.

Rewriting categories for Lean. In the general case, however, we do not have explicit access to $\mathscr{R}^*$ and hence consider subcategories of the tractable subcategory $\mathscr{R}_{\mathcal{T}} \subseteq \mathscr{R}^*$ from Example 3.

Definition 6. A *(Lean) rewriting category* is a subcategory $\mathscr{R} \subseteq \mathscr{R}_{\mathcal{T}}$ of $\mathscr{R}_{\mathcal{T}}$ with $\mathrm{ob}(\mathscr{R}) = \mathsf{Stmt}$.

Of particular interest are rewriting categories $\mathscr{R}$ that restrict the allowed set of tactics. For a set $\mathcal{T}' \subseteq \mathcal{T}$, we say that $\mathscr{R}$ is *generated by* $\mathcal{T}'$ if $\mathscr{R}$ is the inclusion-wise minimal rewriting category, such that for all $T, T' \in \mathsf{Stmt}$ and $t \in \mathcal{T}'$ with $\mathcal{L}(T') = t(\mathcal{L}(T))$, there is an arrow $t : T \to T'$ in $\mathscr{R}$.

Each such category encodes a specific, increasingly conservative notion of available transformations. This hierarchy of subcategory approximations of $\mathscr{R}^*$ is used to scaffold our empirical study in Section 5, where different rule sets yield augmentations of varying semantic strength, and the sensitivity of a prover to the choice measures how far its behavior is from the invariance we desire.

With the rewriting category $\mathscr{R}_{\mathcal{T}}$ in place, and its restrictions $\mathscr{R} \subseteq \mathscr{R}_{\mathcal{T}}$, we are ready to define the structural biases expected from formal provers. We begin with the notion of *$\mathscr{R}$-equivariance of the proof distribution*, ensuring that the structures of solutions produced by the prover align with $\mathscr{R}$.

3.3. Proof equivariance

If for a pair of statements $T, T' \in \mathsf{Stmt}$, a tactic $t \in \mathcal{T}$ transforms $\mathcal{L}(T)$ into $\mathcal{L}(T')$, then any proof $\mathbf{t}$ of T' can be lifted to a proof of T by concatenating t at the beginning, i.e., $[t : \mathbf{t}]$. This observation motivates an equivariance property for formal provers p_θ. Formally, this corresponds to transporting distributions along arrows in the rewriting category. Intuitively, the distribution $p_\theta([t : \cdot] \mid T)$ of completions of proofs of T that start with t should match the distribution of proofs for T'. If a prover p_θ meets this condition, improving its performance on T' would automatically translate to increased performance on T, for proofs that start with t. To ensure that this distribution transport is well-defined and unambiguous, we will assume the following:

Assumption 7. Denote by $\mathcal{T}^*$ the space of finite sequences of Lean tactics, and let $\mathbb{D}_+(\mathcal{T}^*)$ be the set of distributions over $\mathcal{T}^*$, assigning non-zero probability to each $\mathbf{t} \in \mathcal{T}^*$. Let $T \in \mathsf{Stmt}$. Given T, p_θ has a non-zero probability of producing any proof $\mathbf{t} \in \mathcal{T}^*$, i.e. $p_\theta(\cdot \mid T) \in \mathbb{D}_+(\mathcal{T}^*)$.

Assumption 7 holds, in principle, for various classes of models, including LLMs, relying on logit-based next-token sampling. The concept of proof transport can now be defined in terms of functors:

Definition 8 (Proof functor). The *(stochastic) proof functor* on $\mathscr{R}$ is the functor $\mathcal{P} : \mathscr{R} \to \mathbf{Set}$ that maps each object $T \in \mathscr{R}$ to $\mathcal{P}(T) = \mathbb{D}_+(\mathcal{T}^*)$, and whose action on arrows $\mathbf{t} : T \to T'$ is defined as:

$$[\mathcal{P}(\mathbf{t})](\nu)(\mathbf{t}') \propto \nu([\mathbf{t} : \mathbf{t}']) \qquad \forall \nu \in \mathbb{D}_+(\mathcal{T}^*), \mathbf{t}' \in \mathcal{T}^*$$

In words, for any arrow $\mathbf{t} : T \to T'$ in $\mathscr{R}$, $\mathcal{P}(\mathbf{t}) : \mathbb{D}_+(\mathcal{T}^*) \to \mathbb{D}_+(\mathcal{T}^*)$ is a function mapping each distribution $\nu \in \mathbb{D}_+(\mathcal{T}^*)$ to its restriction over proofs that start with $\mathbf{t}$. The proof of functoriality of $\mathcal{P}$ is presented in Appendix E.1. Using proof functors, we can formalize the property outlined above:

Definition 9 ($\mathscr{R}$-equivariance of proof distribution). Let $\mathcal{P} : \mathscr{R} \to \mathbf{Set}$ be the proof functor. A prover p_θ is *$\mathscr{R}$-equivariant in proof distribution* if for all $\mathbf{t} : T \to T'$ in $\mathscr{R}$:

$$p_\theta(\cdot \mid T') = \mathcal{P}(\mathbf{t})(p_\theta(\cdot \mid T))$$

Under Assumption 7, this is a well-typed definition. $\mathscr{R}$-equivariant provers, correspond precisely to *natural transformations* $\mathcal{I} \to \mathcal{P}$, which we properly introduce and elaborate on in Appendix B.1. Importantly, a prover satisfying Definition 9 assigns *structurally matched* proofs matching probabilities, allowing for generalization between any pair of problems linked by an arrow in $\mathscr{R}$.

Proof equivariance in the literature. Although Lean proof equivariance has not been formalized previously, some approaches share closely related inductive biases. In particular, next-tactic predictors (Yang et al., 2023; Li et al., 2024; Xin et al., 2025b) operate on intermediate proof states rather than theorem statements, and can satisfy $\mathscr{R}_{\mathcal{T}}$-equivariance. Note, however, that this symmetry breaks when these models rely on a search under fixed budgets (Appendix B.3). Moreover, their multi-step training introduces optimization challenges, often resulting in weaker performance than single-pass provers. At the same time, incorporating such bias into whole-proof generation models would require severe architectural changes.

Consequently, in this work, we focus on an adjacent notion of bias for formal provers, defined by the invariance of the success probability across statements that share the same mathematical content.

3.4. Success invariance

Proof $\mathscr{R}$-equivariance, stated in Section 3.3, requires proof distributions to transfer according to the arrows of a rewriting category $\mathscr{R}$. With the examples from Figure 2 in mind, a less restrictive condition could require the prover to achieve the same success rate on problems *equivalent* w.r.t $\mathscr{R}$:

Definition 10 ($\mathscr{R}$-equivalent statements). We say that statements $T, T' \in \mathscr{R}$ are *$\mathscr{R}$-equivalent*, denoted as $T \sim_{\mathscr{R}} T'$, if $T \leftrightarrows T'$ in $\mathscr{R}$, i.e. there exist some arrows $\mathbf{t} : T \to T'$ and $\mathbf{t}' : T' \to T$.

The motivation behind Definition 10 is the ability to transform valid proofs between T and T' via the arrows of $\mathscr{R}$. We encapsulate the ability of models to perform such transfer by requiring that the probabilities of success of solving $\mathscr{R}$-equivalent statements are equal:

Definition 11 (Success invariance). A prover p_θ is *$\mathscr{R}$-invariant in success probability* if

$$s_\theta(T) \;=\; s_\theta(T') \qquad \text{for every } T \sim_{\mathscr{R}} T'.$$

Setting $\mathscr{R} = \mathscr{R}^*$ recovers the informal notion of success invariance. An $\mathscr{R}$-invariant prover respects the transformations assigned by $\mathscr{R}$ and guarantees that $\mathscr{R}$-equivalent problems are correctly solved with the same probability. Therefore, improvement on solving a specific theorem T directly translates to improvements over the whole equivalence class $[T]_{\mathscr{R}}$.

A hierarchy of invariances. Under $\mathscr{R}^* \supseteq \mathscr{R}_{\mathcal{T}} \supseteq \mathscr{R}$ of Section 3.2, we obtain the implications

$$\underbrace{\mathscr{R}^*\text{-invariance}}_{\text{ideal success invariance}} \implies \mathscr{R}_{\mathcal{T}}\text{-invariance} \implies \mathscr{R}\text{-invariance}$$

This hierarchy formalizes progressively weaker but more tractable notions of invariance. Invariance with respect to the unreachable maximal category $\mathscr{R}^*$ is the strongest, capturing what we ideally want from a prover. Restricting to a tractable subcategory $\mathscr{R} \subseteq \mathscr{R}_{\mathcal{T}}$ is a necessary condition that we can computationally evaluate. Our methodology pursues the latter, and the observed variance can be seen as a lower bound on the error of $\mathscr{R}^*$-invariance.

4. Test-time scaling with rewriting for success invariance

Section 3.4 formalizes success invariance of a prover p_θ with respect to a rewriting category $\mathscr{R} \subseteq \mathscr{R}_{\mathcal{T}}$. We now address how to realize this property in practice. Prior work (Tian et al., 2026) encourages invariance through training-time data augmentation, but is limited to a small set of transformations and requires retraining. Instead, we seek a *test-time* method that, given an already-trained p_θ that may not satisfy success invariance, produces another prover $\bar{p}_\theta$ that is (approximately) $\mathscr{R}$-invariant, without retraining p_θ. Our construction is inspired by sampling and canonicalization strategies for group invariance (Kim et al., 2025; 2026; Kaba et al., 2023; Schmidt and Stober, 2024; Shumaylov et al., 2025; Singhal et al., 2025).

4.1. The rewriting ensemble

We take the rewriting subcategory $\mathscr{R} \subseteq \mathscr{R}_{\mathcal{T}}$ from Section 3.2 as fixed, together with a sampling procedure $\mu(\cdot \mid T, k)$ that, given a theorem statement $T \in \mathsf{Stmt}$ and a number k, returns k theorems $T_1, \ldots, T_k \in \mathsf{Stmt}$ reachable from T via a finite chain of arrows in $\mathscr{R}$. These variants approximate the equivalence class $[T]_{\mathscr{R}}$, and serve as alternative representations of the same underlying problem.

Definition 12 (Rewriting ensemble). Fix a prover p_θ, budget $N \in \mathbb{N}$ and a variant count $K \leq N$. The (K, N)-*rewriting ensemble prover* $\bar{p}^{(K,N)}_{\theta,\mu}$, applied to a statement T, proceeds as follows:

(1) Sample K variants $\{T_1, \ldots, T_K\} \sim \mu(\cdot \mid T, K)$, with the corresponding transitions $T \xrightarrow{\mathbf{r}_i} T_i$.
(2) Run p_θ for $\lfloor N/K \rfloor$ independent attempts on each T_i.
(3) Return any resulting valid proof, verified with Lean, lifted to a proof of T via the arrow $T \xrightarrow{\mathbf{r}_i} T_i$ in $\mathscr{R}$; return failure if no attempt succeeds.

Assuming independence of attempts, the expected success probability of the ensemble is

$$\bar{s}^{(K,N)}_{\theta,\mu}(T) \;=\; 1 - \mathbb{E}\left[\prod_{i=1}^{K}\big(1 - s_\theta(T_i)\big)^{\lfloor N/K \rfloor}\right]$$

Two limiting cases are worth bearing in mind: $K = 1$ collapses the ensemble to running p_θ on a single random variant, and $K = N$ spreads the budget one attempt per variant. Our analysis below concerns how $\bar{s}^{(K,N)}_{\theta,\mu}(T)$ depends on K, and how it compares across $T \sim_{\mathscr{R}} T'$.

4.2. Invariance in the sample-size limit

For the ensemble prover to satisfy the $\mathscr{R}$-invariance of Definition 11, the sampling distribution $\mu(\cdot \mid T, K)$ must exhibit behavior generalizable over the whole equivalence class $[T]_{\mathscr{R}}$. This is not automatic, as following arrows from a statement T can, in general, reach different statements than when starting in $T' \sim_{\mathscr{R}} T$. We hence assume that μ has syntactically invariant support:

Assumption 13 (Sampling coverage). Let $T \sim_{\mathscr{R}} T'$ and let $T'' \in \mathscr{R}$ be a statement. Suppose that for some K, the probability $\mathbb{P}_\mu(T'' \mid T, K)$ of sampling T'' is greater than 0. Then:

$$\lim_{K\to\infty} \mathbb{P}_\mu(T'' \mid T, K) = \lim_{K\to\infty} \mathbb{P}_\mu(T'' \mid T', K) = 1$$

In words, if T'' can be sampled for some theorem T, as the number of samples grows, it will eventually be sampled for any member of the equivalence class $[T]_{\mathscr{R}}$. Assumption 13 holds in two cases of interest: (i) when every state T'' reachable from T in $\mathscr{R}$ is eventually covered by $\mu(\cdot \mid T, K)$, (ii) when the sampler performs *canonicalization* (Kaba et al., 2023) and outputs a fixed representative of $[T]_{\mathscr{R}}$.

Proposition 14 (**Invariance in the sample-size limit).** *Let $n : \mathbb{N} \to \mathbb{N}$ be a function such that $\lfloor n(x)/x \rfloor \to \infty$ as $x \to \infty$. Then, under Assumption 13, for every $T \sim_{\mathscr{R}} T'$:*

$$\lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T) \;=\; \lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T')$$

so the limiting ensemble prover is $\mathscr{R}$-invariant in success probability (Definition 11).

Informally, averaging over enough variants washes out the dependence on the particular statement from an $\mathscr{R}$-equivalence class. The proof is provided in Appendix E.2.

4.3. Monotonicity in the number of rewritings

We now characterize the optimal number of variants of the input statement T, for reciprocal $\mathscr{R}$, regardless of the choice of sampling protocol μ. Assuming an equal prior on $s_\theta(T')$ for all $T' \sim_{\mathscr{R}} T$, spreading the budget across more versions is never worse, in expectation, than committing to fewer:

Proposition 15 (**Monotonicity).** *Let $\mathscr{R}$ be reciprocal and $T \in \mathsf{Stmt}$ be a statement. Assume that for any $T' \sim_{\mathscr{R}} T$, the success probability $s_\theta(T') \in [0,1]$ is an independent random variable sampled from some (prior) distribution $P_{[T]_{\mathscr{R}}}$. Then, for any budget N and divisors $K \le K'$ of N,*

$$\bar{s}_{\theta,\mu}^{(K,N)}(T) \;\le\; \bar{s}_{\theta,\mu}^{(K',N)}(T),$$

with strict inequality whenever $K < K'$ and s_θ is non-constant on the support of $\mu(\cdot \mid T, K) \subseteq [T]_{\mathscr{R}}$.

Corollary 16. *In particular, $\bar{s}_{\theta,\mu}^{(K,N)}(T) \ge \bar{s}_{\theta,\mu}^{(1,N)}(T)$ for every $K \ge 1$ dividing N.*

The proof of Proposition 15 is given in Appendix E.3. Note that whenever $s_\theta(T) > 0$, repeated sampling (Brown et al., 2024) without rewriting also achieves $\bar{s} \to 1$ as $N \to \infty$, a property the ensemble does not improve on asymptotically. The value of the rewriting ensemble is therefore at a finite budget N. Our empirical results in Section 6 instantiate this comparison at budgets of practical interest.

5. Methodology

Motivated by the sensitivity of LLM-based formal provers to specific formulations, and the guarantees from Section 4, we design our experiments to study the following questions:

- **(Q1)** How far from success invariance are the modern LLM-based single-pass formal provers?
- **(Q2)** Can rewriting ensembles improve their robustness, and even increase performance?

We introduce the *miniF2F-rw* dataset, consisting of verified, $\mathscr{R}_{\mathcal{T}}$-equivalent rewritings of problems from miniF2F (Zheng et al., 2022) and use it to test the robustness of leading open-source models. We then confirm the beneficial impact of rewriting ensembles in two settings: *controlled*, using the verified variants from miniF2F-rw, and *test-time*, producing reformulations at inference time.

Online augmentations. Augmenting formal statements is substantially harder than query rewriting in natural language tasks (Ma et al., 2023; Zhou et al., 2023; O'Brien et al., 2024) due to strict syntactic constraints (Appendix C.1). To address this, we devise a three-stage test-time rewriting algorithm, outlined below and detailed in Appendix C.3:

(1) **Sampling.** Rewrites are drawn from the rewriting category $\mathscr{R}$ generated by non-conditional lemmas from Mathlib (The mathlib Community, 2020) supplemented with a small set of simplification tactics.
(2) **Scoring.** Each sampled variant is ranked by a model-based energy function (Kim et al., 2026), based on the model's surprise when processing the input theorem, which acts as a cheap proxy for s_θ.
(3) **Selection.** The top-scoring variants, together with the seed, are passed to the prover.

Benchmarks. We propose miniF2F-rw benchmark, built on top of miniF2F (Zheng et al., 2022). Each of the 488 original problems is paired with 5-15 rewrites, each with a Lean certificate of semantic equivalence, yielding roughly 5700 statements in total. All variants were additionally manually filtered to remove ones that changed mathematical content. For more details, see Appendix C.2. To further demonstrate the potential of test-time ensembles, we use ProofNet (Azerbayev et al., 2023), and Ineq-Comp (Zhao et al., 2025) containing a 'trans' split, designed specifically to test robustness to transformations of its 'seed' problems.

Models. Following Tian et al. (2026), we focus on leading open-source non-reasoning models in our main experiments: DeepSeek-Prover-V2 (Ren et al., 2025), Goedel-Prover-DPO (Lin et al., 2025), and Goedel-Prover-SFT (Lin et al., 2025). Reasoning provers produce thousands of tokens per proof (Ren et al., 2025, Table 3), which is prohibitive here. We include their smaller-scale evaluations in Appendix D.

Evaluation. On miniF2F-rw, we compare each prover in four configurations, under fixed budget k:

- **seed**: all k attempts on the original statement (corresponds to miniF2F benchmark performance).
- **random**: all k attempts on a single variant, drawn uniformly from the pool in miniF2F-rw.
- **controlled**: 8 variants drawn uniformly from the pool; budget split to $\frac{k}{8}$ attempts per variant.
- **test-time**: seed + 7 variants sampled online; budget split to $\frac{k}{8}$ attempts per variant.

Table 1. PASS@k results of selected LLM provers on the miniF2F-rw benchmark. The best result for each prover on each split is highlighted as **bold**, and the second-best is underlined.

		miniF2F-rw-valid			miniF2F-rw-test		
Model	**Variant**	$k = 8$	$k = 32$	$k = 64$	$k = 8$	$k = 32$	$k = 64$
Goedel-Prover-SFT (Lin et al., 2025)	seed	**57.7**	**62.8**	**64.8**	**51.7**	**56.5**	**57.8**
	random	51.6±0.9	58.2±0.9	60.9±1.1	47.4±0.8	53.3±0.9	55.2±1.0
	controlled	54.7±0.2	60.8±0.2	63.4±0.2	50.0±0.1	55.7±0.1	57.4±0.2
	test-time	56.3	62.6	63.6	50.5	56.0	57.6
Goedel-Prover-DPO (Lin et al., 2025)	seed	**58.6**	63.1	64.3	**52.9**	57.7	59.0
	random	53.9±1.0	59.7±1.0	61.6±1.2	49.5±0.9	54.7±1.0	56.4±1.1
	controlled	57.7±0.2	63.4±0.2	**65.4±0.3**	52.6±0.2	**57.9±0.2**	**59.9±0.2**
	test-time	57.7	**63.5**	65.2	49.1	55.5	57.1
DeepSeek-Prover-V2-7B (Ren et al., 2025)	seed	78.4	79.1	79.5	61.7	64.6	66.8
	random	71.1±1.1	74.1±1.1	75.3±1.2	59.7±0.9	63.2±1.0	64.8±1.1
	controlled	77.2±0.3	79.7±0.3	80.8±0.4	**64.0±0.3**	**67.3±0.3**	**68.8±0.3**
	test-time	**79.7**	**81.1**	**81.5**	62.6	66.0	67.6

Table 2. The comparison between test-time ensembles and baselines on the considered datasets.

Model	**Budget**	**ProofNet-test**	**Ineq-Comp-seed**	**Ineq-Comp-trans**	**Ineq-Comp-ratio**
EvolProver (Tian et al., 2026)	32	–	52.2	34.0	65.2%
Goedel-Prover-DPO (Lin et al., 2025)	32	13.6	40.5	12.3	30.4%
	64	14.5	44.0	15.3	34.8%
Goedel-Prover-DPO + Ensemble	32	15.6	42.3	14.5	34.3%
	64	16.8	46.6	17.0	36.5%
DeepSeek-Prover-V2-7B (Ren et al., 2025)	32	21.9	64.8	34.8	53.7%
	64	23.1	**66.7**	37.3	55.9%
DeepSeek-Prover-V2-7B + Ensemble	32	21.9	63.4	40.8	64.4%
	64	**23.2**	64.7	**45.5**	**70.3%**

We report PASS@k at three budgets, $k \in \{8, 32, 64\}$. Each variant is attempted $n = 64$ times. As the results for **random** and **controlled** depend on the drawn variants, we sample 20,000 selections and report the mean and standard deviation. An ablation on ensemble size is in Appendix C.4.

We further evaluate **seed** and **test-time** mode for chosen provers on ProofNet (Azerbayev et al., 2023) and Ineq-Comp (Zhao et al., 2025). For the latter, in addition to PASS@k, we also measure the ratio of PASS@k results on the 'trans' and 'seed' splits, displaying provers' generalizability to problems with similar proof structures (Zhao et al., 2025; Tian et al., 2026). We compare their results against EvolProver (Tian et al., 2026), which proposed training data augmentations for implicitly learning generalization over equivalent formulations during training time.

6. Evaluation results

Success invariance. Results on miniF2F-rw (Table 1) reveal a consistent drop in performance under **random** perturbations relative to the **seed** baseline *for all tested models*, confirming their sensitivity to input formulation. Interestingly, this degradation is substantially larger on the validation split. For example, at PASS@8, DeepSeek-Prover-V2 drops by 7.3 points on the validation split, compared to only 2.0 on test. This reveals its dependence on surface structures witnessed during model selection.

Rewriting ensembles effectively mitigate this gap. Both **controlled** and **test-time** variants significantly outperform single random rewrites (e.g., +4% PASS@64 for DeepSeek on test), while additionally reducing variance. Notably, despite their bias toward original statements, both Goedel-Prover-DPO and DeepSeek-Prover-V2 benefit from ensembling, surpassing the **seed** baseline. This empirically validates the theoretical advantages outlined in Section 4. Remarkably, DeepSeek's ensemble is competitive with or surpasses reasoning at matched token counts (Appendix D.2).

Test-time ensembling. We compare **seed** baselines and **test-time** ensembles across benchmarks (Table 2), using EvolProver as a retraining-based reference. Despite models' bias toward seed formulations in ProofNet-test and Ineq-Comp-seed, test-time ensembling consistently yields competitive or improved performance over the baselines. On the main split of interest, Ineq-Comp-trans, rewriting ensemble increases PASS@64 of DeepSeek-Prover-V2 from 37.3% to 45.5%, establishing a new state of the art among non-reasoning models. The resulting robustness ratio is competitive with EvolProver (65.2% vs 64.4% at PASS@32). This shows that our framework can efficiently reduce sensitivity of underlying provers to surface forms, without retraining.

Across all datasets, we observe cases where DeepSeek-Prover fails on the original formulation yet consistently solves rewritten variants (Figure 4). In several examples, rewriting increases the success rate from $0/64$ on seed to $8/16$, or even $16/16$, on the reformulation. These results show that our method discovers non-trivial rewrites that substantially improve solvability across diverse domains.

miniF2F

Seed — 0/64 SR

```
theorem mathd_algebra_362 (a b : ℝ)
  (h₀ : a ^ 2 * b ^ 3 = 32 / 27)
  (h₁ : a / b ^ 3 = 27 / 4)
  : a + b = 8 / 3 := by
```

Augmented — 2/16 SR

```
theorem mathd_algebra_362 (a b : ℝ)
  (h₀ : a ^ 2 * b ^ 3 = 32 / 27)
  (h₁ : b ^ 3 / a = 4 / 27)
  : a + b = 8 / 3 := by
```

Tactic used:

```
rw [div_eq_div_iff_comm] at h₁
```

ProofNet

Seed — 0/64 SR

```
theorem exercise_1_1_17
  {G : Type*} [Group G] {x : G}
  {n : ℕ} (hxn : orderOf x = n)
  : x^-1 = x^(n-1 : ℤ) := by
```

Augmented — 16/16 SR

```
theorem exercise_1_1_17
  {G : Type*} [Group G] {x : G}
  {n : ℕ} (hxn : orderOf x = n)
  : x^-1 = x^n * x^-1 := by
```

Tactic used:

```
rw [zpow_sub_one, zpow_coe_nat]
```

Ineq-Comp

Seed — 0/64 SR

```
theorem cauchy_p19 (x y z : ℝ)
  (hx : x > 0) (hy : y > 0) (hz : z > 0)
  (h : 1/(1 + x^2) + 1/(1 + y^2) + 1/(1 + z^2) = 2)
  : x^2 + y^2 + z^2 + 3 ≥ (x + y + z)^2 := by
```

Augmented — 8/16 SR

```
theorem cauchy_p19 (x y z : ℝ)
  (hx : 0 < x) (hy : 0 < y) (hz : 0 < z)
  (h : (1 + x^2)⁻¹ + (1 + y^2)⁻¹ + (1 + z^2)⁻¹ = 2)
  : (x + y + z)^2 ≤ x^2 + y^2 + z^2 + 3 := by
```

Tactic used:

```
norm_num at *
```

Figure 4. Examples of problems solved by the test-time ensemble framework, where seed failed.

7. Related work

Statement augmentations. To evaluate the generalization of Lean provers across related problems, Zhao et al. (2025) introduces Ineq-Comp, a benchmark of inequality questions and their algebraic transformations. While not equivalent under our definition, these variants share the proof structure, testing compositional reasoning. To address the observed robustness gap, EvolProver (Tian et al., 2026) augments training data with fixed abstract syntax tree (AST) transformations. FormalEvolve (Lu et al., 2026) shows that enforcing generation of diverse variants of formal statements helps autoformalization (Zhang et al., 2026; Cabral et al., 2026).

Equivalence of formal statements. Determining equivalence between formal theorems is a core challenge in autoformalization, where reliable evaluation requires accurately verifying equivalence with the ground truth. BEq (Liu et al., 2025), a standard metric, defines equivalence via reducibility: T is equivalent to T' if T can be proved using T', and vice versa. In contrast, we adopt a perspective based on co-reachability within rewriting categories.

Structural representations in formal theorem proving. Prior work has explored explicit structural encodings of formal statements (Paliwal et al., 2020; Li et al., 2021; Blaauwbroek et al., 2024), yet typically representing them as AST-based graphs processed with graph neural networks. While these approaches naturally capture symmetries such as variable renaming and commutativity, they have struggled to scale to harder problems compared to LLM-based methods. To bridge this gap, Algebraic Positional Encodings (APE) (Kogkalidis et al., 2024a) incorporate algebraic structure, e.g., of binary trees, directly into the attention mechanisms. In (Kogkalidis et al., 2024b), they were applied to theorem proving by encoding inputs as ASTs.

Symmetries in deep learning. Symmetry is usually modeled as group actions (Bronstein et al., 2021) and incorporated into neural networks either via weight sharing (Ravanbakhsh et al., 2017; Kondor and Trivedi, 2018), data augmentation (Chen et al., 2020; Lyle et al., 2020), or test-time methods (Kim et al., 2025; 2026; Kaba et al., 2023; Schmidt and Stober, 2024; Shumaylov et al., 2025; Singhal et al., 2025). Prior works generalized to categories via monad algebras (Gavranović et al., 2024) or naturality (Maruyama, 2025), although focusing on theorizing weight sharing. Bottou (2024) posited that rewrite rules of natural language (Harris, 1968; 1991) may form a groupoid, a category with all arrows invertible, though not necessarily preserving semantics. We extend these views, modeling semantics-preserving rewrites with a category, using naturality to formulate invariance and equivariance.

8. Summary, limitations, and future work

We frame Lean theorem proving in a category-theoretic framework, modeling statement augmentations as arrows in rewriting categories. This perspective yields two symmetry principles for LLM-based provers: success invariance and proof equivariance. We show that state-of-the-art provers violate these properties, exhibiting strong sensitivity to equivalent reformulations. To address this, we propose a test-time ensembling method that approximates invariance for arbitrary provers. We provide theoretical guarantees and show improved robustness and performance across benchmarks.

We restrict our main experiments with ensembles to non-reasoning provers. Although we expect the qualitative picture to carry over to reasoning models, the *magnitude* of the achieved gains is left as an open question. Another natural extension is to increase the ensemble expressivity, replacing the currently used model-based energy function with a selector trained to predict proof success, or an adaptive bandit allocating compute across variants at inference. A further idea is to incorporate our symmetry framework into model design, e.g., by extending APE (Kogkalidis et al., 2024a) to categorical structures. Lastly, outside the mathematical domain, rewriting categories could be used to model symmetries of formal languages in programming and logic (Pei et al., 2024; Chang et al., 2026) (see Appendix B.5), as well as certain aspects of natural language (Irpan et al., 2025; Berglund et al., 2024; Kusner et al., 2017; Pres et al., 2026).

Acknowledgments

The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work. https://doi.org/10.5281/zenodo.22558. This work was also supported in part by the National Research Foundation of Korea (RS-2024-00351212) funded by the Korean government (MSIT). Finally, we would like to thank Rob Cornish for helpful discussions.

Impact Statement

The work presented in this paper aims to improve the formal mathematical reasoning skills for machine learning models. Given the specific topic and concrete applications to the mathematical domain, although advancing this field may lead to various potential long-term societal consequences, we do not feel any should yet be highlighted.

References

Haoyu Zhao, Yihan Geng, Shange Tang, Yong Lin, Bohan Lyu, Hongzhou Lin, Chi Jin, and Sanjeev Arora. Ineqcomp: Benchmarking human-intuitive compositional reasoning in automated theorem proving on inequalities. In *Advances in Neural Information Processing Systems*, 2025.

Yuchen Tian, Ruiyuan Huang, WANG XUANWU, Jing Ma, Zengfeng Huang, Ziyang Luo, Hongzhan Lin, Da Zheng, and Lun Du. Evolprover: Advancing automated theorem proving by evolving formalized problems via symmetry and difficulty. In *The Fourteenth International Conference on Learning Representations*, 2026.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Alex Irpan, Alexander Matt Turner, Mark Kurzeja, David K Elson, and Rohin Shah. Consistency training helps stop sycophancy and jailbreaks. *arXiv preprint arXiv:2510.27062*, 2025.

Lukas Berglund, Meg Tong, Maximilian Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. The reversal curse: LLMs trained on "a is b" fail to learn "b is a". In *The Twelfth International Conference on Learning Representations*, 2024.

Zhaofeng Wu, Michihiro Yasunaga, Andrew Cohen, Yoon Kim, Asli Celikyilmaz, and Marjan Ghazvininejad. rewordbench: Benchmarking and improving the robustness of reward models with transformed inputs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

Anjiang Wei, Jiannan Cao, Ran Li, Hongyu Chen, Yuhui Zhang, Ziheng Wang, Yuan Liu, Thiago SFX Teixeira, Diyi Yang, Ke Wang, et al. Equibench: Benchmarking large language models' reasoning about program semantics via equivalence checking. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

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

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

Huajian Xin, Z.Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z.F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction*, 2021.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Yang Li, Dong Du, Linfeng Song, Chen Li, Weikang Wang, Tao Yang, and Haitao Mi. Hunyuanprover: A scalable data synthesis framework and guided tree search for automated theorem proving. *arXiv preprint arXiv:2412.20735*, 2024.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Ming Ding. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. In *Proceedings of the 63rd*

Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2025b.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia LI, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source automated theorem proving. In *Second Conference on Language Modeling*, 2025.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In *The Fourteenth International Conference on Learning Representations*, 2026.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *http://arxiv.org/abs/2504.11354*, 2025.

Jinwoo Kim, Olga Zaghen, Ayhan Suleymanzade, Youngmin Ryou, and Seunghoon Hong. Revisiting random walks for learning on graphs. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jinwoo Kim, Sékou-Oumar Kaba, Jiyun Park, Seunghoon Hong, and Siamak Ravanbakhsh. Inverting data transformations via diffusion sampling. *arXiv preprint arXiv:2602.08267*, 2026.

Sékou-Oumar Kaba, Arnab Kumar Mondal, Yan Zhang, Yoshua Bengio, and Siamak Ravanbakhsh. Equivariance with learned canonicalization functions. In *Fortieth International Conference on Machine Learning*, 2023.

Johann Schmidt and Sebastian Stober. Tilt your head: Activating the hidden spatial-invariance of classifiers. In *Forty-first International Conference on Machine Learning*, 2024.

Zakhar Shumaylov, Peter Zaika, James Rowbottom, Ferdia Sherry, Melanie Weber, and Carola-Bibiane Schönlieb. Lie algebra canonicalization: Equivariant neural operators under arbitrary lie groups. In *The Thirteenth International Conference on Learning Representations*, 2025.

Utkarsh Singhal, Ryan Feng, Stella X. Yu, and Atul Prakash. Test-time canonicalization by foundation models for robust perception. In *Forty-second International Conference on Machine Learning*, 2025.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *The Tenth International Conference on Learning Representations*, 2022.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Kyle O'Brien, Nathan Hoyen Ng, Isha Puri, Jorge Mendez-Mendez, Hamid Palangi, Yoon Kim, Marzyeh Ghassemi, and Thomas Hartvigsen. Improving black-box robustness with in-context rewriting. *Transactions on Machine Learning Research*, 2024.

The mathlib Community. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2020.

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

Haijian Lu, Wei Wang, and Jing Liu. Formalevolve: Neuro-symbolic evolutionary search for diverse and prover-effective autoformalization. *arXiv preprint arXiv:2603.19828*, 2026.

Meiru Zhang, Philipp Borchert, Milan Gritta, and Gerasimos Lampouras. DRIFT: Decompose, retrieve, illustrate, then formalize theorems. In *The Fourteenth International Conference on Learning Representations*, 2026.

Rafael Medeiros Cabral, Tuan Manh Do, Yu Xuejun, Wai Ming Tai, Zijin Feng, and SHEN XIN. Proofflow: A dependency graph approach to faithful proof autoformalization. In *The Fourteenth International Conference on Learning Representations*, 2026.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In *The Thirteenth International Conference on Learning Representations*, 2025.

Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Zhaoyu Li, Binghong Chen, and Xujie Si. Graph contrastive pre-training for effective theorem reasoning. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

Lasse Blaauwbroek, Miroslav Olšák, Jason Rute, Fidel Ivan Schaposnik Massolo, Jelle Piepenbrock, and Vasily Pestun. Graph2tac: online representation learning of formal math concepts. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Konstantinos Kogkalidis, Jean-Philippe Bernardy, and Vikas Garg. Algebraic positional encodings. In *Advances in Neural Information Processing Systems*, 2024a.

Konstantinos Kogkalidis, Orestis Melkonian, and Jean-Philippe Bernardy. Learning structure-aware representations of dependent types. In *Advances in Neural Information Processing Systems*, 2024b.

Siamak Ravanbakhsh, Jeff Schneider, and Barnabás Póczos. Equivariance through parameter-sharing. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Shuxiao Chen, Edgar Dobriban, and Jane H Lee. A group-theoretic framework for data augmentation. *Journal of Machine Learning Research*, 2020.

Clare Lyle, Mark van der Wilk, Marta Kwiatkowska, Yarin Gal, and Benjamin Bloem-Reddy. On the benefits of invariance in neural networks, 2020.

Bruno Gavranović, Paul Lessard, Andrew Joseph Dudzik, Tamara von Glehn, João Guilherme Madeira Araújo, and Petar Veličković. Position: Categorical deep learning is an algebraic theory of all architectures. In *Forty-first International Conference on Machine Learning*, 2024.

Yoshihiro Maruyama. Categorical equivariant deep learning: Category-equivariant neural networks and universal approximation theorems. *arXiv preprint arXiv:2511.18417*, 2025.

Léon Bottou. Conceptual challenges in modern ai, 2024. URL `https://youtu.be/MEeqneHqukg?si=p46WJqu9aPWwVPTR`.

Zellig Harris. Mathematical structures of language. *Interscience tracts in pure and applied mathematics*, 1968.

Zellig Harris. *A theory of language and information: a mathematical approach*. Oxford University Press, 1991.

Kexin Pei, Weichen Li, Qirui Jin, Shuyang Liu, Scott Geng, Lorenzo Cavallaro, Junfeng Yang, and Suman Jana. Exploiting code symmetries for learning program semantics. In *Forty-first International Conference on Machine Learning*, 2024.

Hoyeon Chang, Bálint Mucsányi, and Seong Joon Oh. Dynamics reveals structure: Challenging the linear propagation assumption. In *ICLR 2026 Workshop on Unifying Concept Representation Learning*, 2026.

Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. *Advances in Neural Information Processing Systems*, 2017.

Itamar Pres, Belinda Z Li, Laura Ruis, Zifan Carl Guo, Keya Hu, Mehul Damani, Isha Puri, Ekdeep Singh Lubana, and Jacob Andreas. Position: It's time to optimize for self-consistency, 2026.

Tom Leinster. *Basic category theory*. Cambridge University Press, 2014.

Glynn Winskel. *The formal semantics of programming languages: an introduction*. MIT press, 1993.

Herbert B Enderton. *A mathematical introduction to logic*. Elsevier, 2001.

Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge university press Cambridge, 1989.

Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical society*, 1953.

A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 1937.

Ronald V Book and Friedrich Otto. *String-rewriting systems*. Springer-Verlag, 1993.

Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1998.

Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. Equality saturation: a new approach to optimization. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2009.

Justin Lubin, Jeremy Ferguson, Kevin Ye, Jacob Yim, and Sarah E Chasins. Equivalence by canonicalization for synthesis-backed refactoring. *Proceedings of the ACM on Programming Languages*, 2024.

Marcus Rossel. *An Equality Saturation Tactic for Lean*. PhD thesis, Master's thesis, 2024.

OpenAI. Introducing gpt-5.2. https://openai.com/index/introducing-gpt-5-4/, 2026. Accessed: 2026-04-26.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

A. Background on category theory

Our formalization uses a small amount of elementary category theory. We leave a minimal introduction here, and refer the reader to Leinster (2014) for further background.

A.1. Categories

Categories provide a natural abstraction of local, non-invertible transformations, just as groups provide a natural abstraction of context-independent (i.e. global) and invertible symmetries.

Definition 1 A *category* $\mathscr{C}$ consists of

- a collection of *objects* $\mathrm{ob}(\mathscr{C})$,
- for each $A, B \in \mathrm{ob}(\mathscr{C})$, a collection $\mathscr{C}(A, B)$ of *maps* or *arrows* from A to B,
- for each pair of arrows $f \in \mathscr{C}(A, B)$ and $g \in \mathscr{C}(B, C)$, their *composition* $g \circ f \in \mathscr{C}(A, C)$,
- for each object $A \in \mathrm{ob}(\mathscr{C})$, an element 1_A of $\mathscr{C}(A, A)$, called the *identity* on A,

satisfying the following axioms:

- *associativity*: $(h \circ g) \circ f = h \circ (g \circ f)$ for any $f \in \mathscr{C}(A, B)$, $g \in \mathscr{C}(B, C)$ and $h \in \mathscr{C}(C, D)$,
- *identity laws*: for each $f \in \mathscr{C}(A, B)$, we have $f \circ 1_A = f = 1_B \circ f$.

Example 2 There is a category $\mathbf{Set}$ whose objects are sets, whose maps are ordinary functions, with function composition as composition and identity functions as identities.

Given Example 2 above, it may be tempting to think of maps as functions, but the definition only requires that they compose associatively with identities:

Example 17. Any group G is a category $\mathbf{B}G$ with a single object $\bullet$ and arrows $\mathbf{B}G(\bullet, \bullet) = G$, where composition is the group operation and the identity arrow is the group identity. The invertibility of group elements translates to the statement that every arrow of $\mathbf{B}G$ is an isomorphism.

A.2. Functors

To relate categories to one another and, in particular, to let an abstract category act on concrete data, we use structure-preserving maps between categories, called *functors*.

Definition 4 A functor $F : \mathscr{C} \to \mathscr{D}$ consists of an object map $F : \mathrm{ob}(\mathscr{C}) \to \mathrm{ob}(\mathscr{D})$, and, for each pair $A, B \in \mathrm{ob}(\mathscr{C})$, an assignment on arrows, $F : \mathscr{C}(A, B) \to \mathscr{D}(F(A), F(B))$, satisfying:

- *preservation of composition*: $F(g \circ f) = F(g) \circ F(f)$ whenever $A \xrightarrow{f} B \xrightarrow{g} C$ in $\mathscr{C}$,
- *preservation of identities*: $F(1_A) = 1_{F(A)}$ for every $A \in \mathscr{C}$.

The target $\mathscr{D}$ may differ substantially from the source $\mathscr{C}$, but the compositional structure of $\mathscr{C}$ is reflected in its image under F. Of particular importance in our setting are functors into $\mathbf{Set}$, which model how the abstract transformations in $\mathscr{C}$ *act on* concrete data. We illustrate this with groups:

Example 18. A group action of G on a set X is exactly a functor $F : \mathbf{B}G \to \mathbf{Set}$ sending the unique object $\bullet$ to X and each group element $g \in G$ to a function $F(g) : X \to X$. The functor axioms unfold to the familiar identities $F(gh) = F(g) \circ F(h)$ and $F(1) = 1_X$, which are precisely the defining axioms of a group action.

We will also find useful a simple *implementation functor*, mapping theorems from rewriting categories to the corresponding singleton sets:

Example 5. There is an implementation functor $\mathcal{I} : \mathscr{R}_{\mathcal{T}} \to \mathbf{Set}$ mapping each theorem $T \in \mathsf{Stmt}$ to the singleton set $\{T\}$, while arrows are mapped to the unique functions between singletons.

A.3. Natural transformations

We now introduce natural transformations, the categorical analogue of invariant and equivariant maps. Just as an equivariant map is defined between two G-sets (sets equipped with an action of a group G), a natural transformation is defined between two functors sharing a common source and target.

Definition 19. Let $F, G : \mathscr{C} \to \mathscr{D}$ be two functors. A natural transformation $\alpha : F \to G$ is an object-indexed family of arrows $(\alpha_A : F(A) \to G(A))_{A \in \mathrm{ob}(\mathscr{C})}$ in $\mathscr{D}$ such that, for every arrow $f : A \to B$ in $\mathscr{C}$, the following *naturality square* commutes:

$$\begin{tikzcd} F(A) \arrow[r, "F(f)"] \arrow[d, "\alpha_A"'] & F(B) \arrow[d, "\alpha_B"] \\ G(A) \arrow[r, "G(f)"'] & G(B) \end{tikzcd}$$

Equivalently, $\alpha_B \circ F(f) = G(f) \circ \alpha_A$ for every arrow $f : A \to B$.

The commutativity of this square, called *naturality*, says that the family α is compatible with the transformations encoded by $\mathscr{C}$: transporting along $\mathscr{C}$ and then applying α yields the same result as applying α and then transporting along the corresponding arrow of $\mathscr{D}$. This compatibility is exactly the content of invariance and equivariance in our setting.

Example 20. Taking $\mathscr{C} = \mathbf{B}G$, a natural transformation $\alpha : F \to H$ between two group actions $F, H : \mathbf{B}G \to \mathbf{Set}$ reduces to a single map $\alpha_\bullet : F(\bullet) \to H(\bullet)$ whose naturality square, instantiated at each $g \in G$, becomes the familiar equivariance condition $\alpha_\bullet(g \cdot x) = g \cdot \alpha_\bullet(x)$. When H acts trivially on the target (i.e. $H(g) = 1 \ \forall g \in G$), this reduces further to $\alpha(g \cdot x) = \alpha(x)$, recovering invariance.

The example also illustrates the global character of group transformations: because $\mathbf{B}G$ has a single object, a natural transformation collapses to one map $\alpha_\bullet$ that applies uniformly. In a category with many objects, such as the rewriting categories (introduced in Section 3.2), each object A carries its own component α_A, and naturality is what coordinates them across the arrows of $\mathscr{C}$. This is how the framework captures local, context-dependent transformations central to our setting.

B. Extended formalization of rewriting categories

B.1. Proof equivariance as natural transformations

Let $\mathscr{R}$ be a rewriting category. Recall the definition of the proof functor on $\mathscr{R}$ from Section 3.3:

Definition 8. The *(stochastic) proof functor* on $\mathscr{R}$ is the functor $\mathcal{P} : \mathscr{R} \to \mathbf{Set}$ that maps each object $T \in \mathscr{R}$ to $\mathcal{P}(T) = \mathbb{D}_+(\mathcal{T}^*)$, and whose action on arrows $\mathbf{t} : T \to T'$ is defined as:

$$[\mathcal{P}(\mathbf{t})](\nu)(\mathbf{t}') \propto \nu([\mathbf{t} : \mathbf{t}']) \qquad \forall \nu \in \mathbb{D}_+(\mathcal{T}^*), \mathbf{t}' \in \mathcal{T}^*$$

We then formulate an inductive bias for formal theorem provers in terms of the proof functor $\mathcal{P}$, requiring that their proof distributions can be transported over arrows of $\mathscr{R}$ by the action of $\mathcal{P}$:

Definition 9. Let $\mathcal{P} : \mathscr{R} \to \mathbf{Set}$ be the proof functor. A prover p_θ is *$\mathscr{R}$-equivariant in proof distribution* (with respect to $\mathcal{P}$) if

$$p_\theta(\cdot \mid T') = \mathcal{P}(\mathbf{t})(p_\theta(\cdot \mid T)) \qquad \text{for every arrow } \mathbf{t} : T \to T' \text{ in } \mathscr{R}.$$

For Definition 9 to be well-defined, as $\mathcal{P}(\mathbf{t})$ is a function $\mathbb{D}_+(\mathcal{T}^*) \to \mathbb{D}_+(\mathcal{T}^*)$, we need to assume that $p_\theta(\cdot \mid T) \in \mathbb{D}_+(\mathcal{T}^*)$ for every theorem statement $T \in \mathscr{R}$. This is not a radical simplification, as it is satisfied by whole-proof generation LLMs, relying on logits for next token generation.

Assumption 7. For any $T \in \mathsf{Stmt}$, the prover p_θ satisfies $p_\theta(\cdot \mid T) \in \mathbb{D}_+(\mathcal{T}^*)$.

Let $\mathcal{I}$ be the implementation functor for $\mathscr{R}$. We can treat p_θ as a family $((p_\theta)_T)_{T \in \mathscr{R}}$ of functions $(p_\theta)_T : \{T\} \to \mathbb{D}_+(\mathcal{T}^*)$ mapping $((p_\theta)_T)(T) = p_\theta(\cdot \mid T)$. With this perspective, let $\mathbf{t} : T \to T'$ be an arrow in $\mathscr{R}$. We can rewrite the sides of the equality from Definition 9 as:

$$\begin{aligned} p_\theta(\cdot \mid T') &= ((p_\theta)_{T'})(T') \\ &= ((p_\theta)_{T'})(\mathcal{I}(\mathbf{t})(T)) \\ &= ((p_\theta)_{T'} \circ \mathcal{I}(\mathbf{t}))(T) \end{aligned}$$

Similarly,

$$\mathcal{P}(\mathbf{t})(p_\theta(\cdot \mid T)) = \mathcal{P}(\mathbf{t})(((p_\theta)_T)(T)) = (\mathcal{P}(\mathbf{t}) \circ (p_\theta)_T)(T)$$

which lets us rephrase the condition for $\mathscr{R}$-equivariance in proof distribution as:

$$((p_\theta)_{T'} \circ \mathcal{I}(\mathbf{t}))(T) = (\mathcal{P}(\mathbf{t}) \circ (p_\theta)_T)(T) \qquad \text{for every arrow } \mathbf{t} : T \to T' \text{ in } \mathscr{R}.$$

Both functions $(p_\theta)_{T'} \circ \mathcal{I}(\mathbf{t})$ and $\mathcal{P}(\mathbf{t}) \circ (p_\theta)_T$ are of the type $\{T\} \to \mathbb{D}_+(\mathcal{T}^*)$, so them being equal on the argument T translates to the equality of the functions themselves. As a result, we can simplify the requirement above into:

$$((p_\theta)_{T'} \circ \mathcal{I}(\mathbf{t})) = (\mathcal{P}(\mathbf{t}) \circ (p_\theta)_T) \qquad \text{for every arrow } \mathbf{t} : T \to T' \text{ in } \mathscr{R}.$$

But this is precisely the definition of p_θ being a natural transformation $\mathcal{I} \to \mathcal{P}$.

In conclusion, we have shown that both proposed notions of inductive bias for Lean-based formal theorem provers can be precisely characterized as naturality conditions, and hence as natural transformations. This formulation extends the perspective of geometric deep learning (Bronstein et al., 2021), where structural inductive biases are captured via invariance and equivariance under group actions, to a categorical setting. In this broader context, operating over functors $\mathscr{R} \to \mathbf{Set}$, the appropriate analogue of such equivariance (and also invariance, as we show in the next section) is given by naturality. A similar formulation has been proposed by (Maruyama, 2025), but without being grounded in concrete applications. We find that categories are a natural fit for modeling semantics-preserving rewritings in formal languages.

B.2. Success invariance as natural transformations

Recall that given a rewriting category $\mathscr{R}$, we define the notion of prover $\mathscr{R}$-invariance as follows:

Definition 10. We say that statements $T, T' \in \mathscr{R}$ are *$\mathscr{R}$-equivalent*, denoted as $T \sim_{\mathscr{R}} T'$, if $T \leftrightarrows T'$ in $\mathscr{R}$, i.e. there exist some arrows $\mathbf{t} : T \to T'$ and $\mathbf{t}' : T' \to T$.

Definition 11. A prover p_θ is *$\mathscr{R}$-invariant in success probability* if

$$s_\theta(T) \;=\; s_\theta(T') \qquad \text{for every } T \sim_{\mathscr{R}} T'.$$

For the reciprocal $\mathscr{R}$, we can equivalently formalize this definition in terms of natural transformations.

Let us introduce a functor $\mathcal{J}_{\mathscr{R}} : \mathscr{R} \to \mathbf{Set}$ that maps each object $T \in \mathscr{R}$ to the interval $[0, 1] \subset \mathbb{R}$, and every arrow to the identity function $1_{[0,1]}$. Consider the implementation functor $\mathcal{I} : \mathscr{R} \to \mathbf{Set}$.

For any $T \in \mathscr{R}$, we have $\mathcal{I}(T) = \{T\}$ and $\mathcal{J}(T) = [0, 1]$. This creates a perfect setting to define the success probability map s_θ as a family of functions $(\mathcal{I}(T) \to \mathcal{J}(T))_{T \in \mathscr{R}}$. Concretely, for a given prover p_θ, consider functions $\mathcal{S}^\theta_T : \{T\} \to [0, 1]$ given by $\mathcal{S}^\theta_T(T) = s_\theta(T)$. By the observation above, each of these functions is of type $\mathcal{S}^\theta_T : \mathcal{I}(T) \to \mathcal{J}(T)$. This gives rise to a family $\mathcal{S}^\theta$ of functions $(\mathcal{S}^\theta_T : \{T\} \to [0, 1])_{T \in \mathrm{ob}(\mathscr{R})}$. For reciprocal $\mathscr{R}$, naturality of $\mathcal{S}^\theta$ coincides with $\mathscr{R}$-invariance of p_θ:

Claim 21. *When $\mathscr{R}$ is reciprocal, $\mathcal{S}^\theta$ is a natural transformation $\mathcal{I} \to \mathcal{J}$ if and only if the prover p_θ satisfies Definition 11.*

Proof. Recall that in a reciprocal category $\mathscr{R}$, $T \sim_{\mathscr{R}} T'$ if and only if there is some arrow $T \to T'$. Let $\mathbf{t} : T \to T'$ be an arrow in $\mathscr{R}$. The naturality condition from Definition 19 requires:

$$\mathcal{S}^\theta_{T'} \circ \mathcal{I}(\mathbf{t}) = \mathcal{J}(\mathbf{t}) \circ \mathcal{S}^\theta_T$$

Both these functions are from $\mathcal{I}(T) = \{T\}$ to $\mathcal{J}(T') = [0, 1]$, so the functions above are equal if and only if they act equally on T. Finally, unfolding their definitions, we get:

$$\left(\mathcal{S}^\theta_{T'} \circ \mathcal{I}(\mathbf{t})\right)(T) = \mathcal{S}^\theta_{T'}(T') = s_\theta(T')$$
$$\left(\mathcal{J}(\mathbf{t}) \circ \mathcal{S}^\theta_T\right)(T) = \mathcal{J}(\mathbf{t})(s_\theta(T)) = s_\theta(T)$$

Combining the observations listed above, we can derive the following equivalence chain:

$$\begin{aligned}
p_\theta \text{ is } \mathscr{R}\text{-invariant} &\iff s_\theta(T) = s_\theta(T') \quad \text{for all } T \sim_{\mathscr{R}} T' \\
&\iff s_\theta(T) = s_\theta(T') \quad \text{whenever there is an edge } \mathbf{t} : T \to T' \\
&\iff \left(\mathcal{S}^\theta_{T'} \circ \mathcal{I}(\mathbf{t})\right)(T) = \left(\mathcal{J}(\mathbf{t}) \circ \mathcal{S}^\theta_T\right)(T) \quad \text{whenever } \mathbf{t} : T \to T' \text{ is in } \mathscr{R} \\
&\iff \mathcal{S}^\theta_{T'} \circ \mathcal{I}(\mathbf{t}) = \mathcal{J}(\mathbf{t}) \circ \mathcal{S}^\theta_T \quad \text{whenever } \mathbf{t} : T \to T' \text{ is an arrow in } \mathscr{R} \\
&\iff \mathcal{S}^\theta \text{ is a natural transformation } \mathcal{I} \to \mathcal{J}
\end{aligned}$$

□

Remark 22. More generally, even when $\mathscr{R}$ is not reciprocal, $\mathcal{S}^\theta$ defines a natural transformation $\mathcal{I} \to \mathcal{J}$ provided $s_\theta(T) = s_\theta(T')$ for any statements T, T' connected by an arrow in $\mathscr{R}$. Although this condition does not imply equivalence, an appropriate choice of generating tactics for $\mathscr{R}$ can still induce useful structure in an $\mathscr{R}$-invariant prover. For instance, if the generating set includes tactics invoking external solvers (e.g., `omega`), $\mathscr{R}$-invariance may correspond to the model's ability to reduce statements to forms automatically closed by such tactics.

B.3. Structural biases of next-tactic prediction models

In this work, we study whole-proof LLM-based provers, which take a Lean theorem statement as input and autoregressively generate a complete proof in a single pass. This approach benefits from efficient inference in modern LLMs, but requires committing to an entire reasoning trajectory without intermediate feedback from the Lean compiler.

An alternative approach formulates theorem proving as search over Lean states, with tactics as actions. Next-tactic predictors (Yang et al., 2023; Li et al., 2024; Xin et al., 2025b) condition on the current proof state and predict the next tactic, hence defininig a conditional distribution $m_\theta(\cdot \mid \mathcal{L}(T))$. They are typically integrated into best-first (Li et al., 2024; Xin et al., 2025b) or Monte Carlo tree search (Xin et al., 2025a) frameworks, transforming m_θ into a multi-step prover $M_\theta(\cdot \mid T)$. From the perspective of inductive biases, of particular interest are *sequential* multi-step provers, corresponding to the special case of beam size 1. We will call a multi-step prover M_θ *sequential*, if it repeatedly samples a tactic, applies it to the current state, and continues from the resulting state. The probability of generating a proof $\mathbf{t} = (t_1, \ldots, t_n) \in \mathcal{T}^*$ is then

$$M_\theta(\mathbf{t} \mid T) = \prod_{i=1}^{n} m_\theta(t_i \mid \mathbf{t}_{<i}(\mathcal{L}(T)))$$

By operating on states, these frameworks can implicitly capture the proposed inductive biases:

$\mathscr{R}_\mathcal{T}$ proof equivariance. Let $T \xrightarrow{\mathbf{t}} T'$ be an arrow in $\mathscr{R}_\mathcal{T}$. This corresponds to the relation between induced states: $\mathcal{L}(T') = \mathbf{t}(\mathcal{L}(T))$. Let $\mathcal{P} : \mathscr{R}_\mathcal{T} \to \mathbf{Set}$ be the proof functor. Suppose M_θ is a sequential multi-step prover. Recall that we would call M_θ $\mathscr{R}_\mathcal{T}$-equivariant in proof distribution if:

$$M_\theta(\cdot \mid T') = \mathcal{P}(\mathbf{t})(M_\theta(\cdot \mid T)) \qquad \text{for every arrow } \mathbf{t} : T \to T' \text{ in } \mathscr{R}.$$

Now, let us unwrap the substance of the right-hand side of the equation above. $\mathcal{P}(\mathbf{t})$ is a function that takes a distribution over proofs $\mathcal{T}^*$ and returns the induced marginal distribution of proofs that start with $\mathbf{t}$. Therefore, for any $\mathbf{t}' \in \mathcal{T}^*$, we have:

$$\begin{aligned}
\mathcal{P}(\mathbf{t})(M_\theta(\cdot \mid T))(\mathbf{t}') = \frac{M_\theta([\mathbf{t} : \mathbf{t}'] \mid T)}{M_\theta(\mathbf{t} \mid T)} &= \frac{\prod_{t'_i \in [\mathbf{t}:\mathbf{t}']}^{|[\mathbf{t}:\mathbf{t}']|} m_\theta(t'_i \mid [\mathbf{t} : \mathbf{t}']_{<i}(\mathcal{L}(T)))}{\prod_{t_i \in \mathbf{t}}^{|\mathbf{t}|} m_\theta(t_i \mid \mathbf{t}_{<i}(\mathcal{L}(T)))} \\
&= \prod_{t'_i \in \mathbf{t}'}^{|\mathbf{t}'|} m_\theta(t'_i \mid \mathbf{t}'_{<i}(\mathbf{t}(\mathcal{L}(T)))) \\
&= \prod_{t'_i \in \mathbf{t}'}^{|\mathbf{t}'|} m_\theta(t'_i \mid \mathbf{t}'_{<i}(\mathcal{L}(T'))) \\
&= M_\theta(\mathbf{t}' \mid T')
\end{aligned}$$

Hence, M_θ is $\mathscr{R}$-equivariant in proof distribution. In general, however, when combined with explicit search, this symmetry is lost due to the finite sampling budgets – a state reached once during the search may never be explored, leading to a non-transferable distribution of proofs.

$\mathscr{R}_\iota$-invariance of success probability. The inductive biases of multi-step provers become even clearer when considering success invariance (Section 3.4). Let $\mathscr{R}_\iota$ denote the rewriting category generated solely by the empty tactic sequence $\iota \in \mathcal{T}^*$. Every arrow in $\mathscr{R}_\iota$ is then of the form $T \xrightarrow{\iota} T'$ and satisfies $\mathcal{L}(T') = \iota(\mathcal{L}(T)) = \mathcal{L}(T)$. Hence, whenever $T \leftrightarrows T'$ in $\mathscr{R}_\iota$, both statements compile to the same initial proof state. As a result, M_θ performs search from an identical starting point on both inputs, implying equal success probabilities. Therefore, M_θ is $\mathscr{R}_\iota$-invariant in success probability. Notably, this argument does not require M_θ to be sequential.

Remark 23. Next-tactic prediction introduces a more complex training objective, making optimization challenging and leading to weaker empirical performance. A promising direction for future work is to combine these paradigms by integrating state-conditioned reasoning into single-pass approaches.

B.4. Capturing compositional behavior in Lean proofs

In this work, we consider rewriting categories $\mathscr{R}$ in which each tactic sequence induces a distinct arrow. Concretely, for arrows $\mathbf{t}, \mathbf{t}' : T \to T'$ in $\mathscr{R}$, if the underlying tactic sequences $\mathbf{t}$ and $\mathbf{t}'$ are syntactically distinct, then the corresponding arrows are treated as distinct. Nevertheless, the formalization of the Lean statement–tactic space introduced in Section 3 can capture richer structural properties of this space, requiring provers to reason about the underlying structure of tactics rather than their syntactic form alone.

Consider the tactics t_1, t_2, and t_3 shown in Figure 5. Tactics t_1 and t_2 rewrite the current proof state using the Mathlib lemmas `Nat.sub_add_eq` and `Nat.sub_add_cancel`, respectively, while t_3 applies both these rewriting rules, within a single tactic.

Hence, the sequential composition $[t_1 : t_2]$ has the same effect as applying t_3 directly. Although both induce the same transformation function $d : \Omega \to \Omega$ on Lean states, the concatenation $[t_1 : t_2]$ and t_3 remain syntactically distinct, as shown in Figure 5. A categorical formulation allows these transformations to be identified in $\mathscr{R}$ by requiring that, for any objects $T, T' \in \mathscr{R}$, the morphisms $[t_1 : t_2] : T \to T'$ and $t_3 : T \to T'$ coincide, without imposing any equivalence between t_3 and either t_1 or t_2 individually. This notion of compositional equivalence cannot be naturally expressed using graph- or group-based representations.

Tactic t_1
```
rw [Nat.sub_add_eq]
```

Tactic t_2
```
rw [Nat.sub_add_cancel]
```

Tactic $[t_1 : t_2]$
```
rw [Nat.sub_add_eq]
rw [Nat.sub_add_cancel]
```

Tactic t_3
```
rw [Nat.sub_add_eq, Nat.sub_add_cancel]
```

Figure 5. Composition $[t_1 : t_2]$ has the same effect as t_3, despite them being syntactically different.

The example in Figure 5 concerns tactic sequences that are globally equivalent as functions $\Omega \to \Omega$. Category-theoretic structure also enables a finer notion of local equivalence. Consider the tactics l_1 and l_2 from Figure 6. Both introduce the auxiliary statement that $3x \equiv 1 \pmod 7$, but construct the proof differently. Consequently, they are not extensionally equivalent: for example, l_2 requires the presence of a hypothesis `hx` in the proof state. Nevertheless, categorical structure allows these transformations to be identified whenever both act successfully on a given state, capturing an informal notion of "having proved an auxiliary result inside the main proof, no matter how, the model should continue in the same way".

Tactic l_1
```
have h₀ : x * 3 ≡ 1 [ZMOD 7] := by
  norm_num
```

Tactic l_2
```
have h₀ : x * 3 ≡ 1 [ZMOD 7] := by
  simpa [hx]
```

Figure 6. Tactics l_1 and l_2 may act differently on some statements.

These examples further motivate the use of category theory as a formal framework for the space of Lean statements and tactics. In contrast to graph- or group-based representations, categorical structure naturally captures compositional and contextual equivalences between tactics, including both globally equivalent transformations and locally equivalent behaviors conditioned on proof states. This additional expressive power enables modeling structural relationships between tactics that extend beyond purely syntactic or functional representations.

B.5. Rewriting categories of formal languages

In this subsection, we leave a brief remark on how rewriting categories could be used to model symmetries of formal languages, beyond the mathematical domain that we focus on in this work.

We consider formal languages equipped with a syntax and a semantics, where the syntax specifies which expressions are valid, and the semantics assigns meaning to expressions. Programming and logical languages are the canonical examples (Winskel, 1993; Enderton, 2001; Girard et al., 1989). A central concern therein is checking whether two given expressions are semantically equivalent. In programming languages, this could be if two programs map every input to the same output, or if they execute in the same way. In logic, this could be if each formula is derivable from the other. Unfortunately, these are largely undecidable (Rice, 1953; Turing, 1937).

A workaround is using a tractable notion of equivalence based on a set of rewriting rules that takes an expression and

syntactically transforms it without altering its semantics of interest. If two expressions can be made identical through rewriting, we call them syntactically equivalent, which implies they are semantically equivalent (as in Section 3.4, the converse does not necessarily hold). This approach is usually implemented with *rewriting systems* that reduce expressions to normal forms that cannot be further rewritten, and compare those to check equivalence (Book and Otto, 1993; Baader and Nipkow, 1998). This idea is used in compiler optimization (Tate et al., 2009), program refactoring (Lubin et al., 2024), and (classical) automated theorem proving (Rossel, 2024).

Rewriting categories (Section 3.2) can be used to describe rewriting systems. An (abstract) rewriting system consists of a set A, elements called objects or signatures, together with a binary relation on A, denoted by $\to$, called the rewrite relation (Book and Otto, 1993). Then we can construct a rewriting category $\mathscr{R}$ by

- taking as its objects the set A,
- taking as its arrows the reflexive transitive closure $\xrightarrow{*}$ of $\to$, i.e. the transitive closure of $\to \cup \{\mathrm{id}\}$, where id is the identity relation.

In other words, we can lift a rewriting system to a rewriting category by composing its rewriting relations and identity. Thus, the relationship between them is analogous to that between a group and its generating set. An $\mathscr{R}$-invariant map, defined in naturality sense (as in Appendix B.2), preserves syntactic equivalence by construction, and hence preserves semantic equivalence.

From a rewriting system, one can take a subset of its relations to obtain a restriction. For example, if the relations of a system $(A, \to)$ is an indexed union $\to = \to_1 \cup \to_2$ of two rewriting rules, one can take $(A, \to_1)$ as a restriction. It would yield sound rewrites, but more incomplete as some semantically equivalent expressions would no longer be syntactically equivalent. For the respective categories $\mathscr{R}$ and $\mathscr{R}_1$, this translates to a subcategory relation $\mathscr{R}_1 \subseteq \mathscr{R}$, and so a hierarchy of invariances as in Section 3.4. We exemplify with lambda calculus, a foundational example in programming and logic.

Example 24. The untyped lambda calculus consists of lambda expressions defined by the syntax $e ::= x \mid (\lambda x.e) \mid (e\, e)$ where x ranges over a namespace. Its rewriting system can be identified as $(\Lambda, \to)$, where Λ is the set of all lambda expressions, and the relations $\to = \to_\alpha \cup \to_\beta \cup \to_\eta$ consist of

- α-conversion (variable renaming) that rewrites $\lambda x.M$ to $\lambda y.M[x := y]$ if y is not free in M,
- β-reduction (function application) that reduces an expression of the form $(\lambda x.t)s$ to $t[x := s]$.
- η-conversion (extensionality) that converts between $\lambda x.fx$ and f if x does not appear free in f.

Hence, there is a restricted rewriting system $(\Lambda, \to_\alpha)$ that only performs variable renaming. Between the rewriting categories $\mathscr{R} = (\Lambda, \xrightarrow{*})$ and $\mathscr{R}_\alpha = (\Lambda, \xrightarrow{*}_\alpha)$, this translates to $\mathscr{R}_\alpha \subseteq \mathscr{R}$.

In the context of deep learning, Example 24 illustrates how approaches that aim to analyze or improve the robustness of neural networks on formal languages primarily focusing on variable renaming (Mirzadeh et al., 2025; Pei et al., 2024) yield weaker invariances than more general rewritings, such as in our work.

C. Extended methodology

This section provides additional details on the experimental methodology. We first describe the construction of the miniF2F-rw dataset, which augments miniF2F (Zheng et al., 2022) with equivalent problem reformulations (Appendix C.2). We then detail the sampling mechanism $\mu(\cdot \mid T, K)$ used in the test-time ensemble setting (Section C.3) and report ablations over its design choices (Section C.3) and the number of ensemble variants in miniF2F-rw evaluations (Appendix C.4). The codebase, containing implementations of rewriting ensembles and the miniF2F-rw dataset can be found at: https://github.com/kolejnyy/rw-ensembles.

C.1. Practical challenges of statement reformulation in Lean

Before detailing our methodology, we emphasize that statement rewriting in Lean is substantially more challenging than in natural language settings. In NLP, query rewriting is a standard technique for improving the robustness and reasoning capabilities of LLMs (Ma et al., 2023; Zhou et al., 2023; O'Brien et al., 2024). Such approaches rely on the assumption that semantically equivalent queries admit interchangeable solutions: answers to rewritten queries can typically be reused directly or mapped back to the original formulation with minimal effort. For example, the question "*What is the capital of Germany?*" may be rewritten as "*Which city in Germany is the capital?*", while preserving both meaning and answerability.

Moreover, modern LLMs exhibit strong understanding of natural language, making the generation of such rewrites fairly simple.

In Lean theorem proving, these assumptions no longer apply. Proof correctness is determined by a syntactically strict formal system, where even minor modifications to a statement may invalidate an existing proof. As a result, semantically equivalent statements do not necessarily admit interchangeable proofs, and transferring proofs between related formulations can require non-trivial transformations. Consider the statements T_1 and T_2 in Figure 7. Although `n + 1` and `1 + n` are mathematically equivalent, they are syntactically distinct terms, and Lean does not identify them automatically. Consequently, the proof term `h.symm`, which proves T_1, is rejected as a proof of T_2. This illustrates that meaningfully transforming a problem while remaining capable of recovering a proof of the original is non-trivial.

Statement T_1

```
theorem T1 (n : ℕ) (h : n + 1 = 3) : 3 = n + 1 := by
```

Proof of statement T_1

```
theorem T1 (n : ℕ) (h : n + 1 = 3) : 3 = n + 1 := by
  exact h.symm
```

Statement T_2

```
theorem T2 (n : ℕ) (h : 1 + n = 3) : 3 = n + 1 := by
```

Incorrect proof of statement T_2

```
theorem T2 (n : ℕ) (h : 1 + n = 3) : 3 = n + 1 := by
  exact h.symm
```

Figure 7. Lean proofs do not transfer between equivalent statements.

A sound augmentation mechanism can be implemented in two distinct ways:

- **Out-of-Lean.** Analogously to augmentation methods in natural language tasks, an LLM may generate rewrites of the original statement directly. However, this approach additionally requires constructing a Lean certificate that transfers proofs of the augmented statement back to the original problem. In general, such certificates are non-trivial to obtain and may themselves require LLM inference, which needs to be accounted for under fixed computation budgets.
- **Within-Lean.** Alternatively, augmentations can be generated directly within Lean by exploring the space of proof states via tactics. This avoids using LLM calls outside the core proving process, but introduces two challenges: selecting meaningful rewrite rules without excessively complicating the proof state, and defining a mapping from proof states back to theorem statements, since LLM provers operate on statements, whereas Lean applies tactics on states.

In our test-time sampling framework (Appendix C.3), we adopt the latter approach. We sample transformations from the large collection of unconditional premises available in Mathlib, comprising over 30,000 lemmas. This yields non-trivial statement rewritings, as illustrated in Figure 4, while retaining a principled mechanism for proof transfer through tactic composition.

C.2. Generation of the miniF2F-rw dataset

To systematically demonstrate semantic inconsistency among LLM-based Lean provers, we construct *miniF2F-rw*, a dataset of 5726 theorems (2922 validation, 2804 test) obtained as semantics-preserving reformulations of problems from the miniF2F benchmark. Each variant is paired with Lean certificates of `variant` $\Leftrightarrow$ `seed`, ensuring that any proof of the augmented statement can be translated into a proof of the original, and vice versa. We further apply manual checks to remove variants that introduce spurious complexity or trivialize the task. An overview of the pipeline is shown in Figure 8.

Initial variant generation. First, we generate candidate augmentations by prompting GPT-5.4 to produce 5-15 rewrites of each miniF2F statement. For each seed theorem, we then sample 20% of its variants and apply a second transformation, again using GPT-5.4, to rename variables and hypothesis names. The exact prompts used in this process are listed in Appendix F. At the end of this stage, we obtain over 7500 statements for subsequent verification.

Certifying equivalence. To ensure semantic equivalence between augmented statements and their seeds, we employ a two-stage validation procedure. First, we manually check each variant and discard those that alter the content in an unreasonably complicated (e.g., replacing `1` with `0.factorial`) or simplistic (e.g., simplifying algebraic expressions) way.

Second, we generate Lean certificates establishing `variant` $\Leftrightarrow$ `seed`. Each such implication is decomposed into a series of simpler subproblems. Writing a seed theorem T and its variant T' as:

$$T := (C_1 \wedge \ldots \wedge C_k) \implies G$$
$$T' := (C'_1 \wedge \ldots \wedge C'_l) \implies G'$$

for the direction `variant` $\Rightarrow$ `seed`, it suffices to show:

$$(C_1 \wedge \ldots \wedge C_k) \implies C'_i \qquad \text{for all } i \in 1, \ldots, l$$
$$G' \implies G$$

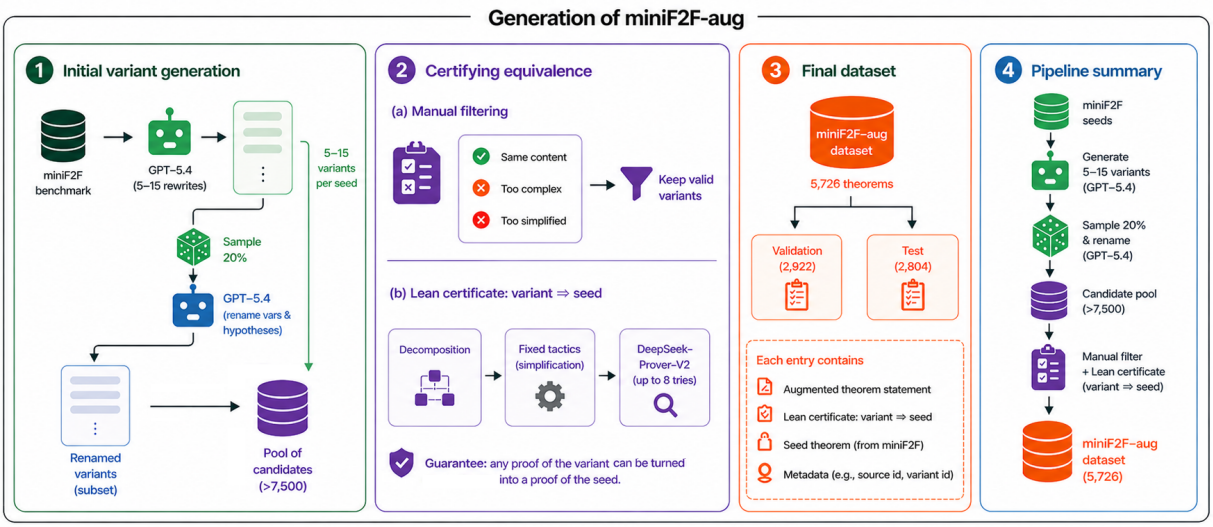

Figure 8. Process of generation of the miniF2F-rw dataset.

Given these components, a proof of T can be obtained by reconstructing the premises of T', invoking T' (simulated via `sorry`), and finally reducing G' to G (see Figure 9 for a code example). The inverse implication can be derived analogously, by flipping T and T'. This methodology resembles that of BEq (Liu et al., 2025), which calls a prover to produce "a proof of T using T'". However, BEq only checks whether the name of T' was mentioned in the proof, without assessing whether it was meaningfully used, while our decomposition enforces semantic correctness.

To close each of these subgoals, we initially apply a set of fixed simplification tactics (see the proof of `goal_to_goal` in Figure 9). If this fails, we invoke DeepSeek-Prover-V2 (Ren et al., 2025) with up to 8 attempts per subproblem. Since the decomposed goals are significantly simpler than the original statements, this budget suffices in most cases. Finally, we manually check the failure instances and repair them when a small number of augmentations of a specific problem are successfully verified.

C.3. Description of the test-time sampling mechanism

Beyond the controlled miniF2F-rw setting, where augmentations are precomputed and verified, we consider a test-time ensemble framework, generating reformulations online. The procedure consists of three stages: rewriting, composition, and reranking, described below and illustrated in Figure 10.

Rewriting tactics sampling. In the first stage, we use Lean to sample rewrite sequences applicable to each expression in the input statement. For a given expression, we construct sequences of $(L_1, \ldots, L_d)$ lemmas drawn from Mathlib (The mathlib Community, 2020). To guarantee semantic equivalence, we restrict attention to the subset $\mathbb{M}_0$ of unconditioned lemmas of the form $P = Q$ or $P \Leftrightarrow Q$, whose application is inherently reversible. Despite this constraint, the resulting transformation space strictly generalizes prior augmentation schemes (Tian et al., 2026; Paliwal et al., 2020; Li et al., 2021; Blaauwbroek et al., 2024), which rely on abstract syntax tree (AST) symmetries to ensure soundness. In contrast, $\mathbb{M}_0$ induces a richer rewriting category $\mathscr{R}_{\mathbb{M}_0}$ that supports non-trivial transformations (see Figure 4), while preserving correctness through Lean tactic semantics. This provides a tractable, more complete approximation to the full tactic rewriting category $\mathscr{R}_{\mathcal{T}}$.

To sample the sequences of rewriting rules, we use a methodology similar to the `rw?` tactic, yet with increased exploration breadth and depth. At each step, candidate lemmas from $\mathbb{M}_0$ are matched against the expression's AST, ranked using the `rw?`'s utility heuristic, and validated via application. We pick the top-b valid rewrites and recursively expand them up to depth d. This yields up to b^d candidate sequences and corresponding rewritten expressions.

To control complexity, we apply AST-based filtering, discarding rewrites whose tree depth or node count exceeds that of the original expression. We then deduplicate syntactically equal outputs arising from different rewrite paths. After validation, we found that $b = 15$ and $d = 2$ provide a favorable performance-to-efficiency trade-off, usually producing 5–50 distinct variants per expression.

```
theorem mathd_algebra_405_v2 (S : Finset ℕ)
  (h : ∀ x, x \in S ↔ 0 < x \and (x + 2) ^ 2 < 20) :
  2 = S.card := by
  sorry

theorem cond_1 (S : Finset ℕ)
  (h : ∀ x, x \in S ↔ 0 < x \and x ^ 2 + 4 * x + 4 < 20)
  : ∀ x, x \in S ↔ 0 < x \and (x + 2) ^ 2 < 20 := by
  intro x
  simp_all only [Finset.mem_filter, Finset.mem_range, Finset.mem_univ,
    true_and_iff, and_true_iff, Nat.cast_ofNat, Nat.cast_one,
    Nat.cast_add, Nat.cast_mul, Nat.cast_sub, Nat.cast_pow]
  constructor <;> intro h <;>
  (try constructor) <;>
  (try simp_all [sq]) <;>
  (try ring_nf at * <;> omega) <;>
  (try nlinarith) <;>
  (try linarith)

theorem goal_to_goal (S : Finset ℕ) (aug_goal : 2 = S.card) : S.card = 2
  := by
  try decide
  try native_decide
  try trivial
  try linarith
  try nlinarith
  try omega
  try norm_num
  try field_simp
  try simp
  try ring_nf
  try ring
  try exact?

theorem mathd_algebra_405 (S : Finset ℕ)
  (h : ∀ x, x \in S ↔ 0 < x \and x ^ 2 + 4 * x + 4 < 20) :
  S.card = 2 := by
  have aug_0 : ∀ x, x \in S ↔ 0 < x \and (x + 2) ^ 2 < 20 := cond_1 S h
  have aug_goal : 2 = S.card := mathd_algebra_405_v2 S aug_0
  exact goal_to_goal S aug_goal
```

Figure 9. An example of a Lean certificate, proving `mathd_algebra_405_v2` ⇒ `mathd_algebra_405`.

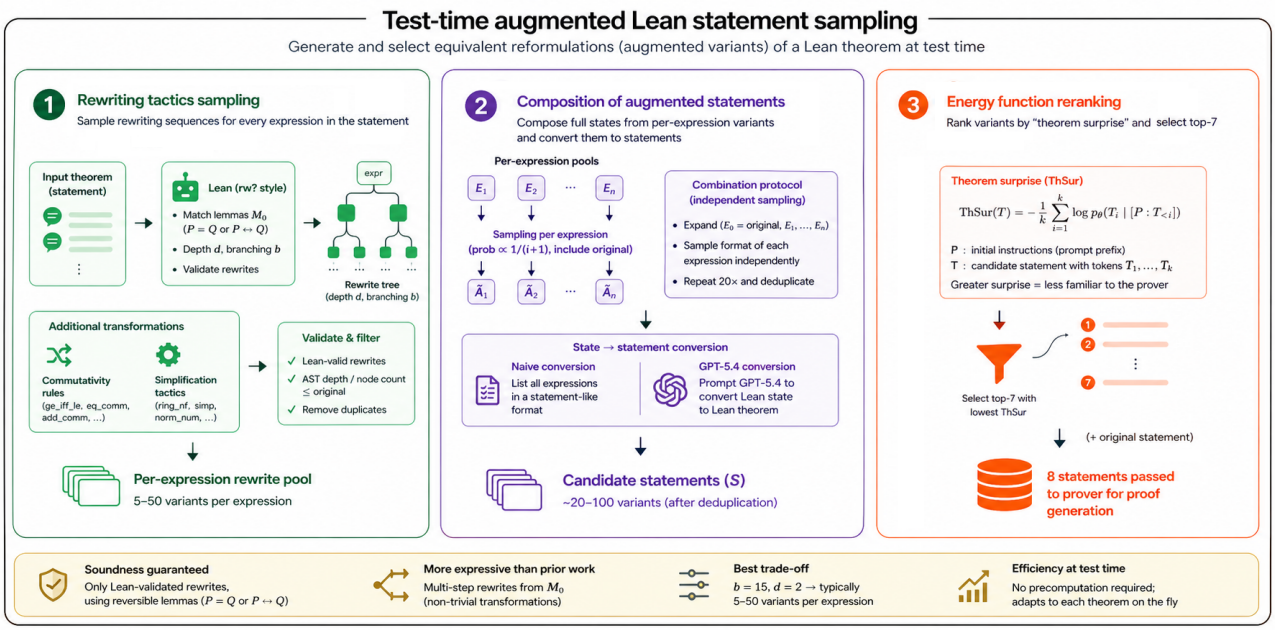

Figure 10. Sampling process for the test-time ensemble mechanism.

As the search space over $\mathbb{M}_0$ remains large, limiting to top-b candidates may omit semantically meaningful rewrites. To mitigate this, we explicitly augment the pool with selected commutativity rules (`ge_iff_le`, `←ge_iff_le`, `←gt_iff_lt`, `gt_iff_lt`, `eq_comm`, `add_comm`, `mul_comm`, `and_comm`, `or_comm`) and standard normalization tactics (**`ring_nf`**, **`simp`**, **`norm_num`**, **`field_simp`**). The resulting outputs are incorporated into the final set of candidate rewrites.

Composition of augmented statements. Given the set of rewritten versions for each expression (premises and goal), we construct augmented theorem statements suitable for downstream provers. This involves two design choices: (i) how to select rewrites for each statement, and (ii) how to map rewritten expressions back to a coherent theorem.

For rewrite selection, a simple baseline modifies a single expression per variant, keeping all others unchanged. While straightforward, this approach becomes restrictive for statements with many components, as all but one remain unchanged in each augmentation. To increase diversity, we instead sample a rewrite for each expression independently from its candidate set. Concretely, for an expression E with ranked rewrites $(E_1, \ldots, E_n)$, we form $(E_0 = E, E_1, \ldots, E_n)$ and sample E_i with probability proportional to $\frac{1}{i+1}$. Applying this procedure across all expressions yields a sampled state. Repeating it 20 times and removing duplicates produces a diverse set of augmented states.

We then convert each state into a theorem statement. This step is non-trivial, as Lean's internal representation may differ from its surface syntax (e.g., implicit type annotations or notation changes such as `Real.pi` vs. π). While syntactic correctness is not strictly required (generated proofs are ultimately checked against the original statement after tactics-based transfer), poorly formed outputs can degrade prover performance. We therefore consider two strategies. A naive approach directly linearizes the state into a statement-like format. A more advanced state-to-statement function is realised by prompting an LLM to convert provided Lean states to theorems. Empirically, the latter yields higher-quality outputs and resolves common inconsistencies, as confirmed by ablations. In our experiments, we use GPT-5.4 (OpenAI, 2026) for this conversion (prompt available in Appendix F.3). This choice is primarily for convenience: the mapping is highly systematic, and a small LLM trained on (state, statement) pairs should suffice. We leave this simplification for future work.

Energy function reranking. Given a set S of retrieved statements, we select a subset of 7 variants to pass to the prover. Inspired by orbit canonicalization methods (Kaba et al., 2023; Singhal et al., 2025; Lubin et al., 2024), we rank candidates using an energy function that approximates the success probability s_θ of prover p_θ. As estimating s_θ directly would require full proof attempts, we introduce a lightweight proxy, called *theorem surprise*.

The underlying intuition is that LLM-based provers perform better on in-distribution inputs. We therefore prioritize statements that appear more "natural" to the model, as measured by their token-level likelihood under p_θ. Concretely, for a statement T, to generate a proof for T, p_θ is prompted with input $[P : T]$, where P is the initial set of instructions ("*You are*

Table 3. Success rate per ensemble style on miniF2F-valid (with DeepSeek-Prover-V2)

Budget	**8**	**16**	**32**	**64**
DSP-V2 (baseline)	78.4	78.8	79.1	79.5
DSP-V2 (ThSur + simp)	78.7	79.2	79.7	80.1
DSP-V2 (ThSur + simp + excomm + comb20)	78.9	79.6	80.1	80.6
DSP-V2 (ThSur + simp + excomm + comb20 + LLM)	**79.7**	**80.5**	**81.1**	**81.5**

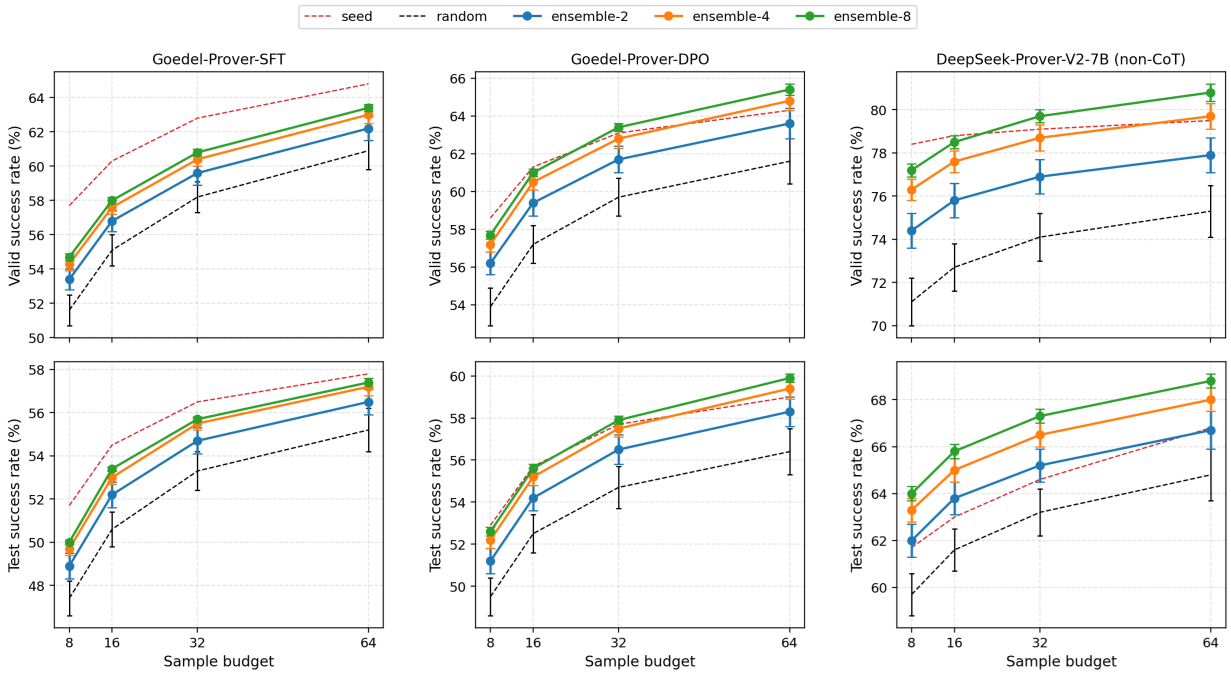

Figure 11. PASS@k success rates on miniF2F-rw for ensemble sizes 2, 4, and 8 (solid lines), with single seed and random-sample baselines as narrower dashed lines. Ensemble and random curves include $\pm$ one estimated standard error.

an expert in Lean4..."). Letting $T_1, \ldots, T_l$ be the tokens representing T, we define the theorem surprise of T as:

$$\text{ThSur}(T) = -\frac{1}{l}\sum_{i=1}^{l} \log p_\theta(T_i \mid [P : T_{<i}])$$

where $p_\theta(T_i \mid [P : T_{<i}])$ is the probability of p_θ generating token T_i after seeing P and the first $i-1$ tokens of T. Higher values indicate less familiar inputs. We rank candidates by ThSur and select the seven lowest-scoring statements, which, together with the original problem, are passed to p_θ for proof generation.

Ablation on the sampling protocol. We iteratively refine the sampling procedure via ablations on miniF2F-valid (Table 3). All variants use theorem surprise for reranking and include rewrites obtained from standard simplification tactics. Augmenting the rewrite pool with explicit commutativity rules and switching to the combination-based state construction (sampling 20 variants) yields consistent improvements. Finally, replacing the naive state-to-statement conversion with GPT-5.4 further boosts performance, highlighting the importance of well-formed input representations for prover success.

C.4. Ablation on the number of sampled rewrites

Proposition 15 predicts that ensemble performance should increase with the number of sampled augmentations. We empirically validate this by evaluating ensemble sizes of 2, 4, and 8 on miniF2F-rw. Figure 11 reports validation and test success rates as a function of the sample budget for each non-reasoning prover in our main setup. Across all models, PASS@k improves monotonically with the ensemble size, consistent with the theoretical prediction. Notably, for Goedel-Prover-DPO and DeepSeek-Prover-V2, using 4 variants already surpasses the seed baseline despite the performance gap

between seed and random augmentations, with further gains observed at 8 samples. Based on these results, we use 8 variants in all miniF2F-rw experiments.

C.5. Adaptation of PASS@k to ensembles

In its standard formulation, given C successes over N attempts, PASS@k is measured as:

$$\text{PASS@}k = 1 - \frac{\binom{N-C}{k}}{\binom{N}{k}}$$

representing the probability of seeing at least one success across randomly chosen k. For an ensemble, running on K variants with success-total attempt pairs $(N_1, C_1), \ldots, (N_K, C_K)$, we report:

$$\text{PASS}_{ens}\text{@}k = 1 - \prod_{i=1}^{K} \frac{\binom{N_i - C_i}{k/K}}{\binom{N_i}{k/K}}$$

C.6. Testing environment and hyperparameters.

Hardware specifications. We perform proof generation on a machines equipped with Intel Platinum 8628 CPUs and NVIDIA H100 GPUs (80GB VRAM). Each model is allocated 8 CPU cores, 64GB RAM, and a single H100 GPU. Under this setup, non-reasoning models generate one proof attempt every 0.5-1.3s on average, while models with explicit informal reasoning require 7-10s per attempt.

Proof verification is conducted on machines with Intel Platinum 8628 or Intel Xeon Gold 6326 CPUs. This stage is highly parallelizable; we dispatch separate jobs per problem, each using 8 CPU cores and 64GB RAM. Verification latency is approximately 1s per attempt. Overall, for the miniF2F-rw benchmark (5,700 problems, 64 attempts each), a single non-reasoning prover requires approximately 4–5 GPU-days for generation and an additional ~4 CPU-days for Lean-based verification.

Model hyperparameters. All LLM provers used in our experiments are prompted with the original prompts from their repositories. We use vLLM (Kwon et al., 2023) for more efficient generation, and sampling temperature of 0.7 for all models. The maximum number of output tokens is set to 2000 for non-reasoning, and 8192 for reasoning provers. For calls to GPT-5.4 (OpenAI, 2026), we fix the temperature at 0.2.

API costs. We use GPT-5.4 to augment problem statements during construction of the miniF2F-rw dataset. The cost per statement ranges from $\$0.01$ to $\$0.05$, totaling approximately $\$14.50$ for 488 benchmark problems. At test time, GPT-5.4 is additionally employed to convert states into statements for ensembling. This incurs a total cost of $\$18.19$ across all datasets (miniF2F, ProofNet, and Ineq-Comp), corresponding to approximately $\$0.009$ per problem. As discussed in Appendix C.3, this transformation can be implemented using a local model, mitigating cost in practical deployments.

Table 4. PASS@k results of selected reasoning LLM provers on the miniF2F-rw benchmark.

		minif2f-aug-valid				*minif2f-aug-test*			
Model	**Variant**	$k=1$	$k=8$	$k=16$	$k=32$	$k=1$	$k=8$	$k=16$	$k=32$
Goedel-Prover-V2-8B	seed	**68.8**	**81.1**	**82.5**	**83.2**	**58.9**	74.6	76.4	77.5
	random	65.2$\pm$0.7	79.5$\pm$0.6	81.2$\pm$0.6	82.2$\pm$0.6	56.0$\pm$0.7	72.9$\pm$0.7	75.3$\pm$0.8	76.7$\pm$0.8
	controlled	–	80.6$\pm$0.1	81.9$\pm$0.1	82.8$\pm$0.1	–	**75.0$\pm$0.1**	**77.0$\pm$0.1**	78.3$\pm$0.1
	test-time	–	80.4	81.9	82.8	–	74.4	76.7	**78.6**
DeepSeek-Prover-V2-7B	seed	**67.4**	**78.1**	**79.7**	80.7	**58.0**	**71.0**	**73.2**	**75.0**
	random	62.7$\pm$0.8	75.9$\pm$0.7	77.7$\pm$0.8	79.3$\pm$1.0	53.4$\pm$0.8	68.5$\pm$0.7	70.8$\pm$0.7	73.0$\pm$0.9
	controlled	–	77.7$\pm$0.1	79.5$\pm$0.2	**81.0$\pm$0.2**	–	70.3$\pm$0.1	72.3$\pm$0.1	74.2$\pm$0.1
	test-time	–	77.5	79.3	**81.0**	–	70.9	72.9	74.2
Kimina-Prover-Distill-8B	seed	**61.9**	73.4	75.4	76.6	**55.0**	68.2	70.0	71.3
	random	57.6$\pm$0.9	70.9$\pm$0.9	73.6$\pm$0.9	75.8$\pm$1.1	52.2$\pm$0.8	66.5$\pm$0.8	68.9$\pm$0.8	70.7$\pm$0.9
	controlled	–	**74.2$\pm$0.2**	**76.5$\pm$0.2**	**78.3$\pm$0.2**	–	68.8$\pm$0.1	71.0$\pm$0.1	72.5$\pm$0.1
	test-time	–	72.0	74.5	76.3	–	**70.1**	**72.6**	**74.4**

Table 5. Average token usage per inference mode per attempt for DeepSeek-Prover-V2-7B.

Model	**miniF2F-valid**	**miniF2F-test**
DeepSeek-Prover-V2-7B	346	395
DeepSeek-Prover-V2-7B (CoT)	3138	3763

D. Additional reasoning models experiments

D.1. Robustness of reasoning provers on miniF2F-rw

We evaluate three state-of-the-art open-source reasoning LLM provers: Goedel-Prover-V2-8B (Lin et al., 2026), Kimina-Prover-Distill-8B (Wang et al., 2025) and DeepSeek-Prover-V2-7B (Ren et al., 2025) (in CoT mode). We run each prover for $n = 32$ attempts on all variants from miniF2F-rw and report PASS@k for $k \in \{1, 8, 16, 32\}$.

The results are presented in Table 4. Although the degradation is less severe than for non-reasoning provers (Table 1), suggesting that an explicit reasoning phase improves robustness to formal rewrites, all models still consistently prefer the seed statements over their augmented variants. For example, on the miniF2F-rw-test split, Goedel-Prover-V2-8B drops from 58.9 to 56.0 PASS@1 and from 77.5 to 76.7 PASS@8. The effect is more pronounced for DeepSeek-Prover-V2-7B, whose PASS@1 declines from 67.4 to 62.7 on the validation split, and from 58.0 to 53.4 on the test set. This indicates that current reasoning-oriented provers remain sensitive to syntactic formulation and do not exhibit full invariance under semantics-preserving transformations. Consequently, these results further motivate the structural bias criteria proposed in this work, even for reasoning models.

Consistent with the observations in Section 6, ensembling across semantically equivalent variants reduces performance variance and consistently improves over selecting a single random sample. This trend holds across all evaluated models, datasets, and sampling budgets. Importantly, both the **controlled** and **test-time** sampling protocols remain at least competitive with the **seed** baselines in every configuration. For Kimina-Prover-Distill-8B, they substantially outperform the default model, increasing PASS@32 on the test split from 71.3% to 74.4%. Overall, these results suggest that, even for reasoning-based formal provers, allocating the inference budget across multiple equivalent representations of the input theorem can yield significant performance gains.

D.2. Comparisons at matched token counts on miniF2F

Table 5 contains average token counts per problem on miniF2F. Since reasoning mode uses 9-10$\times$ more tokens on average per problem, in Tables 1 and 4, a comparison at roughly matched token counts within DeepSeek-Prover-V2 results would be comparing reasoning with budget k against ensemble with budget $8k$. In these comparisons, ensemble is competitive with and often surpasses reasoning, showing its utility as an inference-time scaling method in this setting. While this matches token counts, we remark that in terms of the computational cost of forward passes, reasoning would be more inefficient, because in modern LLMs, generating a single long chain is worse than a set of shorter generations due to lack of parallelizability and the quadratic cost of self-attention.

E. Proofs

E.1. Functoriality of the proof functor

Let $\mathscr{R}$ be a rewriting category. Recall the definition of the (stochastic) proof functor map on $\mathscr{R}$:

Definition 8. Let $\mathbb{D}_+(\mathcal{T}^*)$ be the set of all distributions over the set $\mathcal{T}^*$ of Lean proofs, which assign non-zero probability to each element $\mathbf{t} \in \mathcal{T}^*$. The *(stochastic) proof functor* on $\mathscr{R}$ is the functor $\mathcal{P} : \mathscr{R} \to \mathbf{Set}$ that maps each object $T \in \mathsf{Stmt}$ to $\mathcal{P}(T) = \mathbb{D}_+(\mathcal{T}^*)$, and whose action on arrows $T \xrightarrow{\mathbf{t}} T'$ of $\mathscr{R}$ is defined as:

$$\mathcal{P}(\mathbf{t})(\nu)(\mathbf{t}') \propto \nu([\mathbf{t}\,{:}\,\mathbf{t}']) \qquad \forall \nu \in \mathbb{D}_+(\mathcal{T}^*), \mathbf{t}' \in \mathcal{T}^*$$

We claim that the so-defined map is indeed a functor.

Proposition 25. *The map from Definition 8 is a well-defined functor $\mathcal{P} : \mathscr{R} \to \mathbf{Set}$.*

Proof. First of all, note that $\mathbb{D}_+(\mathcal{T}^*) \in \mathbf{Set}$, so the function of $\mathcal{P}$ on objects is well-defined.

Pick any $\mathbf{t}, \mathbf{t}' \in \mathcal{T}^*$ and $\nu \in \mathbb{D}_+(\mathcal{T}^*)$. The concatenation $[\mathbf{t} : \mathbf{t}']$ is also an element of $\mathcal{T}^*$. Moreover, by positivity of ν, the sum of $\nu([\mathbf{t} : \mathbf{t}''])$ over $\mathbf{t}'' \in \mathcal{T}^*$ is greater than 0. Hence, we can define a distribution $\nu' \in \mathbb{D}_+(\mathcal{T}^*)$ with probabilities proportional to $\nu([\mathbf{t} : \cdot])$, as follows:

$$\nu'(\mathbf{t}') = \frac{\nu([\mathbf{t} : \mathbf{t}'])}{\sum_{\mathbf{t}'' \in \mathcal{T}^*} \nu([\mathbf{t} : \mathbf{t}''])} \qquad \forall\, \mathbf{t}' \in \mathcal{T}^*$$

being precisely the distribution $\mathcal{P}(\mathbf{t})(\nu)(\mathbf{t}')$ from Definition 8. Therefore, the action of $\mathcal{P}$ on arrows of $\mathscr{R}$ is also well-defined. It now suffices to show that $\mathcal{P}$ satisfies the functor conditions, i.e. preserves:

- *identities*: Let $T \in \mathsf{Stmt}$ be a statement. The identity arrow 1_T in $\mathscr{R}$ is associated with the application of the empty tactic sequence ι. Unfolding the definition from above, for all $\nu \in \mathbb{D}_+(\mathcal{T}^*), \mathbf{t}' \in \mathcal{T}^*$:

$$\begin{aligned}\mathcal{P}(1_T)(\nu)(\mathbf{t}') &= \frac{\nu([\iota : \mathbf{t}'])}{\sum_{\mathbf{t}'' \in \mathcal{T}^*} \nu([\iota : \mathbf{t}''])} \\ &= \nu([\iota : \mathbf{t}']) \\ &= \nu(\mathbf{t}')\end{aligned}$$

  meaning that $\mathcal{P}(1_T)(\nu) = \nu$ for all $\nu \in \mathbb{D}_+(\mathcal{T}^*)$. Consequently, $\mathcal{P}(1_T) = 1_{\mathbb{D}_+(\mathcal{T}^*)} = 1_{\mathcal{P}(T)}$.

- *composition*: Let $T \xrightarrow{\mathbf{t}} T' \xrightarrow{\mathbf{t}'} T''$ be arrows in $\mathscr{R}$. We need to show $\mathcal{P}(\mathbf{t}') \circ \mathcal{P}(\mathbf{t}) = \mathcal{P}(\mathbf{t}' \circ \mathbf{t})$. Let $\nu \in \mathbb{D}_+(\mathcal{T}^*)$ be a distribution and let $\mathbf{t}'' \in \mathcal{T}^*$ be a proof. Then:

$$\begin{aligned}(\mathcal{P}(\mathbf{t}') \circ \mathcal{P}(\mathbf{t}))\,(\nu)(\mathbf{t}'') &= (\mathcal{P}(\mathbf{t}'))\,(\mathcal{P}(\mathbf{t})(\nu))(\mathbf{t}'') \\ &\propto (\mathcal{P}(\mathbf{t})(\nu))([\mathbf{t}' : \mathbf{t}'']) \\ &\propto \nu([\mathbf{t} : \mathbf{t}' : \mathbf{t}''])\end{aligned}$$

  and similarly

$$\mathcal{P}(\mathbf{t}' \circ \mathbf{t})(\nu)(\mathbf{t}'') \propto \nu([(\mathbf{t}' \circ \mathbf{t}) : \mathbf{t}'']) = \nu([\mathbf{t} : \mathbf{t}' : \mathbf{t}''])$$

  Since both $(\mathcal{P}(\mathbf{t}') \circ \mathcal{P}(\mathbf{t}))(\nu)$ and $\mathcal{P}(\mathbf{t}' \circ \mathbf{t})(\nu)$ are distributions over $\mathcal{T}^*$, by the result above, they must be equal. As ν was arbitrary, we can conclude that $\mathcal{P}(\mathbf{t}') \circ \mathcal{P}(\mathbf{t}) = \mathcal{P}(\mathbf{t}' \circ \mathbf{t})$. □

E.2. Proof of Proposition 14

To prove $\mathscr{R}$-invariance of the ensemble framework in the limit of samples, we assume the following:

Assumption 13. Let $T \sim_{\mathscr{R}} T'$ and let $T'' \in \mathscr{R}$ be a statement. Suppose that for some K, the probability $\mathbb{P}_\mu(T'' \mid T, K)$ of sampling T'' is greater than 0. Then:

$$\lim_{K \to \infty} \mathbb{P}_\mu(T'' \mid T, K) = \lim_{K \to \infty} \mathbb{P}_\mu(T'' \mid T', K) = 1$$

As mentioned in Section 4.2, this assumption is true whenever the sampling protocol explores the space of reachable states thoroughly enough, or when it canonicalizes the input theorem, transforming it into a specific representative of its equivalence class. In our methodology described in Section 5, we propose a mixture of these ideas: sampling various statements reachable from the input theorem, followed by re-ranking using an energy function. This way, we approximate canonicalization by optimizing an objective function over a subspace of reachable reformulations.

Under Assumption 13, we can prove that the ensemble framework is $\mathscr{R}$-invariant in success probability in the limit of computational budget and the number of samples:

Proposition 14 Let $n : \mathbb{N} \to \mathbb{N}$ be a function such that $\lfloor n(x)/x \rfloor \to \infty$ as $x \to \infty$. Then, for every $T \sim_{\mathscr{R}} T'$:

$$\lim_{K\to\infty} \bar{s}^{(K,n(K))}_{\theta,\mu}(T) \;=\; \lim_{K\to\infty} \bar{s}^{(K,n(K))}_{\theta,\mu}(T')$$

so limiting ensemble prover is $\mathscr{R}$-invariant in success probability.

Proof. Recall that the probability $\bar{s}^{(K,N)}_{\theta,\mu}(T)$ of success of an (N, K)-ensemble on statement T is:

$$\bar{s}^{(K,N)}_{\theta,\mu}(T) \;=\; 1 - \mathbb{E}\left[\prod_{i=1}^{K}\big(1 - s_\theta(T_i)\big)^{\lfloor N/K \rfloor}\right]. \tag{1}$$

Let $T \sim_{\mathscr{R}} T'$ be $\mathscr{R}$-equivalent statements. Denote by $\mathbb{P}_\mu(T'' \mid T, K)$ the probability that T'' will be among the samples drafted by $\mu(\cdot \mid T, K)$. Suppose there exists some T'' with $s_\theta(T'') > 0$, such that for some K, $\mathbb{P}_\mu(T'' \mid T, K) > 0$. Then, by the Assumption 13, we have:

$$\lim_{K\to\infty} \mathbb{P}_\mu(T'' \mid T, K) = \lim_{K\to\infty} \mathbb{P}_\mu(T'' \mid T', K) = 1$$

Since each term $\big(1 - s_\theta(T_i)\big)^{\lfloor N/K \rfloor}$ in Equation (1) is bounded above by 1, we can write:

$$\begin{aligned}\mathbb{E}_{(T_1,\ldots,T_K)\,\sim\,\mu(\cdot|T,K)}\left[\prod_{i=1}^{K}\big(1 - s_\theta(T_i)\big)^{\lfloor N/K \rfloor}\right] \quad &\leq \quad \mathbb{P}_\mu(T'' \mid T, K) \cdot \big(1 - s_\theta(T'')\big)^{\lfloor N/K \rfloor} \\ &\quad + \big(1 - \mathbb{P}_\mu(T'' \mid T, K)\big)\end{aligned}$$

Indeed, if T'' appears among the sampled states, which happens with probability $\mathbb{P}_\mu(T'' \mid T, K)$, the product is at most $\big(1 - s_\theta(T'')\big)^{\lfloor N/K \rfloor}$. Otherwise, with the probability of $1 - \mathbb{P}_\mu(T'' \mid T, K)$, we can crudely bound it by 1.

With this observation, we can derive a lower bound for the limit from the problem statement:

$$\begin{aligned}\lim_{K\to\infty} \bar{s}^{(K,n(K))}_{\theta,\mu}(T) &= \lim_{K\to\infty}\left(1 - \mathbb{E}_{(T_1,\ldots,T_K)\,\sim\,\mu(\cdot|T,K)}\left[\prod_{i=1}^{K}\big(1 - s_\theta(T_i)\big)^{\lfloor n(K)/K \rfloor}\right]\right) \\ &= 1 - \lim_{K\to\infty}\left(\mathbb{E}_{(T_1,\ldots,T_K)\,\sim\,\mu(\cdot|T,K)}\left[\prod_{i=1}^{K}\big(1 - s_\theta(T_i)\big)^{\lfloor n(K)/K \rfloor}\right]\right) \\ &\geq 1 - \lim_{K\to\infty}\Big(\mathbb{P}_\mu(T'' \mid T, K) \cdot \big(1 - s_\theta(T'')\big)^{\lfloor n(K)/K \rfloor} + \big(1 - \mathbb{P}_\mu(T'' \mid T, K)\big)\Big)\end{aligned}$$

Now, we already know that

$$\lim_{K\to\infty} \mathbb{P}_\mu(T'' \mid T, K) = 1$$

Furthermore, as $n(K)/K \to \infty$ as $K \to \infty$ and $0 \leq 1 - s_\theta(T'') < 1$, we can deduce that

$$\lim_{K\to\infty} \big(1 - s_\theta(T'')\big)^{\lfloor n(K)/K \rfloor} = 0$$

Combining these observations, we can derive that

$$\begin{aligned}\lim_{K\to\infty} \bar{s}^{(K,n(K))}_{\theta,\mu}(T) &\geq 1 - \lim_{K\to\infty}\Big(\mathbb{P}_\mu(T'' \mid T, K) \cdot \big(1 - s_\theta(T'')\big)^{\lfloor n(K)/K \rfloor} + \big(1 - \mathbb{P}_\mu(T'' \mid T, K)\big)\Big) \\ &= 1 - (1 \cdot 0 + (1 - 1)) \\ &= 1\end{aligned}$$

As by definition, $\lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T) \leq 1$, we must have $\lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T) = 1$. Since the proof of this property did not involve any properties specific to T apart from the fact that $\lim_{K\to\infty} \mathbb{P}_\mu(T'' \mid T, K) = 1$, using $\lim_{K\to\infty} \mathbb{P}_\mu(T'' \mid T', K) = 1$, we can derive an analogous result for T', giving us:

$$\lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T) \;=\; \lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T') = 1$$

For the second case, suppose that for all T'' with $\mathbb{P}_\mu(T'' \mid T, K) > 0$ for some K, the probability $s_\theta(T'')$ that the prover p_θ solves T'' equals 0. This clearly leads to:

$$\bar{s}_{\theta,\mu}^{(K,n(K))}(T) \;=\; 0 \qquad \text{for all } K$$

and consequently

$$\lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T) \;=\; 0$$

Then, an analogous result has to hold for $\lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T')$. Indeed, otherwise, there would exist some statement S with $s_\theta(S) > 0$ such that $\mathbb{P}_\mu(S \mid T', K) > 0$ for some K. By Assumption 13, since $T' \sim_{\mathscr{R}} T$, this would imply that

$$\lim_{K\to\infty} \mathbb{P}_\mu(S \mid T, K) = \lim_{K\to\infty} \mathbb{P}_\mu(S \mid T', K) = 1$$

so in particular, there would exist some value of K such that $\mathbb{P}_\mu(S \mid T, K) > 0$. This contradicts the assumption that $\mu(\cdot \mid T, K)$ cannot sample any statement with positive provability chance s_θ. Hence,

$$\lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T) \;=\; \lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T') = 0$$

Either way,

$$\lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T) \;=\; \lim_{K\to\infty} \bar{s}_{\theta,\mu}^{(K,n(K))}(T')$$

□

E.3. Proof of Proposition 15

As the second theoretical result about the proposed ensembling framework, we establish that under minor symmetry assumptions on the success rate of equivalent theorems, increasing the number of considered reformulations improves the chances of finding a correct proof.

The assumption used for this proposition states that for a given prover p_θ, statement T and a rewriting category $\mathscr{R}$, the probability $s_\theta(T')$ of producing a correct proof for $T' \in [T]_{\mathscr{R}}$ can be viewed as a random variable sampled from a distribution $P_{[T]_{\mathscr{R}}}$, characteristic for the equivalence class $[T]_{\mathscr{R}}$. Intuitively, this means that given two equivalent formulations of the same problem, we cannot predict which one of them will be more favorable to p_θ. This is motivated by the empirical examples of inconsistency of modern formal provers (Figure 2, Section 6).

Before proving the main theorem, we first establish a helpful auxiliary result:

Lemma 26. *Let X be a $[0,1]$-valued random variable, and let $K \leq K'$ be divisors of N. Then,*

$$\mathbb{E}\Big[\, X^{\lfloor N/K' \rfloor} \Big]^{K'} \leq \mathbb{E}\Big[\, X^{\lfloor N/K \rfloor} \Big]^{K}$$

Proof. If $K = K'$, we trivially get equality, so suppose $K < K'$. Let $a = N/K'$ and $b = N/K$. Then, $a < b$. Applying Hölder's inequality with exponents $r = \frac{b}{a} = \frac{K'}{K} > 1$ and $s = \frac{b}{b-a} > 1$, gives:

$$\mathbb{E}[X^a] = \mathbb{E}[X^a \cdot 1^b] \leq (\mathbb{E}[(X^a)^r])^{\frac{1}{r}} \cdot (\mathbb{E}[(1^b)^s])^{\frac{1}{s}} = (\mathbb{E}[X^b])^{\frac{a}{b}}$$

Therefore, raising both sides to the power K', we obtain:

$$\mathbb{E}\Big[\, X^{\lfloor N/K' \rfloor} \Big]^{K'} = (\mathbb{E}[X^a])^{K'} \leq \big((\mathbb{E}[X^b])^{\frac{a}{b}}\big)^{K'} = \mathbb{E}\Big[\, X^{\lfloor N/K \rfloor} \Big]^{\frac{a}{b}\cdot K'} = \mathbb{E}\Big[\, X^{\lfloor N/K \rfloor} \Big]^{K}$$

□

Proposition 15 Let $\mathscr{R}$ be reciprocal, and $T \in \mathscr{R}$ be a statement. Assume that for any $T' \sim_{\mathscr{R}} T$, the success probability $s_\theta(T') \in [0,1]$ can be viewed as an independent random variable sampled from some (prior) distribution $P_{[T]_{\mathscr{R}}}$. Then, for any budget N and divisors $K \leq K'$ of N,

$$\bar{s}^{(K,N)}_{\theta,\mu}(T) \;\leq\; \bar{s}^{(K',N)}_{\theta,\mu}(T),$$

with strict inequality whenever s_θ is non-constant on the support of $\mu(\cdot \mid T, K) \subseteq [T]_{\mathscr{R}}$.

Proof. Recall that if $\mathscr{R}$ is reciprocal, any statement T' reachable from T by arrows of $\mathscr{R}$ is, in fact, $\mathscr{R}$-equivalent to T. Hence, $\mu(\cdot \mid T, K)$ samples from $[T]_{\mathscr{R}}$. The success rate $s_\theta(T_i)$ of the ith sampled statement can be seen as a random variable S_i following the distribution $S_i \sim P_{[T]_{\mathscr{R}}}$. Moreover, these random variables are independent, so we can rewrite the probability $\bar{s}^{(K,N)}_{\theta,\mu}(T)$ as:

$$\begin{aligned}
\bar{s}^{(K,N)}_{\theta,\mu}(T) &= 1 - \mathbb{E}_{\{T_1,\ldots,T_K\} \sim \mu(\cdot|T,K)} \left[\prod_{i=1}^{K} \big(1 - s_\theta(T_i)\big)^{\lfloor N/K \rfloor} \right] \\
&= 1 - \mathbb{E}_{\{T_1,\ldots,T_K\} \sim \mu(\cdot|T,K)} \left[\prod_{i=1}^{K} \big(1 - S_i\big)^{\lfloor N/K \rfloor} \right] \\
&= 1 - \mathbb{E}_{\{T_1,\ldots,T_K\} \sim \mu(\cdot|T,K)} \left[\mathbb{E} \left[\prod_{i=1}^{K} \big(1 - S_i\big)^{\lfloor N/K \rfloor} \right] \right] \\
&= 1 - \mathbb{E}_{\{T_1,\ldots,T_K\} \sim \mu(\cdot|T,K)} \left[\prod_{i=1}^{K} \mathbb{E}\Big[\big(1 - S_i\big)^{\lfloor N/K \rfloor} \Big] \right] \\
&= 1 - \mathbb{E}_{\{T_1,\ldots,T_K\} \sim \mu(\cdot|T,K)} \left[\prod_{i=1}^{K} \mathbb{E}\Big[\big(1 - S\big)^{\lfloor N/K \rfloor} \Big] \right] \\
&= 1 - \mathbb{E} \left[\prod_{i=1}^{K} \mathbb{E}\Big[\big(1 - S\big)^{\lfloor N/K \rfloor} \Big] \right] \\
&= 1 - \mathbb{E}\Big[\big(1 - S\big)^{\lfloor N/K \rfloor} \Big]^K
\end{aligned}$$

where $S \sim P_{[T]_{\mathscr{R}}}$. Analogously,

$$\bar{s}^{(K',N)}_{\theta,\mu}(T) = 1 - \mathbb{E}\Big[\big(1 - S\big)^{\lfloor N/K' \rfloor} \Big]^{K'}$$

For notational convenience, let $X = 1 - S$. It suffices to show:

$$1 - \mathbb{E}\Big[X^{\lfloor N/K' \rfloor} \Big]^{K'} \geq 1 - \mathbb{E}\Big[X^{\lfloor N/K \rfloor} \Big]^{K}$$

which follows trivially from Lemma 26. Tracing back the argument using Hölder's inequality, we can additionally conclude that the equality holds if random variables $X^{\lfloor N/K \rfloor}$ and 1 are proportional, which is equivalent to requiring X to be constant. This, in turn, translates to s_θ being constant on the support $\mu(\cdot \mid T, K)$. □

F. Prompts

F.1. Rewrite generation prompt for GPT

```
You are an expert Lean4 mathematician.
Task:
Generate {variants_min}-{variants_max} alternative theorem statements that are
    semantically equivalent
to the provided Lean theorem. Produce as many high-quality distinct rewrites as possible,
    up to the maximum.
Hard constraints:
1) Keep the same variables and hypotheses names exactly as in the input.
2) Keep the same theorem name stem and append suffixes `_v2`, `_v3`, ... in increasing
    order.
3) Do not include proofs; every theorem must end exactly at `:= by` with nothing after
(`sorry`, tactics, or extra lines).
4) Each rewrite must remain mathematically equivalent to the original theorem.
5) Avoid trivial whitespace-only edits. Prefer algebraic/logical rewrites.
6) DO NOT substitute hypothesis values/terms into other hypotheses or the goal (e.g. if `b
     = 30`, do not replace `b` with `30`).
7) Do not weaken/strengthen the theorem or add junk conjuncts/disjuncts (forbidden: `\and
    True`, `\iff False`, extra unrelated assumptions/goals).
Diversity requirement:
- Include several SIMPLE rewrites that only use symmetry/commutativity style
    transformations.
- Examples of SIMPLE rewrites:
  * `a = b` -> `b = a`
  * `a < b` -> `b > a`
  * `2 * x = y` -> `y = x * 2`
  * `u + v = c` -> `v + u = c`
  * `x ^ 3` -> `x * x * x`
  * `x ^ 2` -> `x * x`
  * `a * b` -> `b * a`
- Across the whole set, rewrite every hypothesis and the goal at least once when possible.
- Also include non-trivial but equivalent reformulations (not only simple ones).
Coverage checklist (must satisfy across produced variants):
- At least 3 variants must rewrite power notation (`^ 2` or `^ 3`) into repeated
    multiplication.
- At least 3 variants must use explicit symmetry/commutativity in products or equalities.
- At least 2 variants should rewrite division-style equalities into product-style
    equalities when safe.
  Example pattern: `a / b = c` -> `a = c * b` or `a = b * c`.
- At least 2 variants should alter parenthesization/ordering of multiplicative expressions
    .
- At least 2 variants should convert '|' divisibility statements into [MOD] or [ZMOD]
    expressions, and vice versa.
- Do not keep one key hypothesis text unchanged in almost all variants.
Allowed transformations include:
- equivalent rearrangements of equalities/inequalities
- equivalent predicate forms (`a | b` vs congruence/mod-0 forms when appropriate)
- syntactic rewrites using commutativity/associativity/distributivity
- rewriting `0 < x` as `x > 0`, rewriting `eg Even x` as `Odd x` when appropriate
- rewriting equations into equivalent multiplied/divided forms without changing
    assumptions
- replacing expressions with provably equivalent forms
Few-shot examples (from existing miniF2F 50x20 rewrites):
f{examples}
Now produce rewrites for this theorem:
f-- theorem id: {theorem_label}
```lean4
f{formal_statement.strip()}
```
Output format:
- Output ONLY Lean theorem declarations.
- Separate each theorem block with one blank line.
```

```
- Start from suffix `_v2`.
- Do not use markdown code fences.
- Never print `sorry` or a proof body after `:= by`.
```

F.2. Variable renaming prompt for GPT

```
You are an expert Lean 4 assistant.
Task:
Given a single Lean 4 theorem declaration (through `:= by`), produce a
**renaming** of its variables and hypothesis names only. The mathematical
meaning must stay the same (α-equivalence): only binder identifiers change.
Rules:
1) Keep the same theorem name as in the input.
2) Do not change the conclusion or hypothesis **types** except where a name
appears as a binder (you may rename bound variables consistently).
3) `variable_map` maps old variable names → new names (e.g. `x` → `u`).
4) `hypothesis_map` maps old hypothesis names → new names (e.g. `h` → `h1` or `h0`).
5) If you rename nothing in a category, use an empty object `{}` for that map.
6) `original` must be exactly the input theorem string (after trimming).
7) `renamed` must be the full theorem line(s) ending with `:= by` only - no `sorry`,
tactics, or proof body after `by`.
Output format:
- Output **only** one JSON object, no markdown fences, no commentary.
- Schema:
'  {original: ..., renamed: ..., variable_map: {...}, hypothesis_map: {...}}'
f-- theorem id: {label}
Input theorem:```lean4
f{stmt}
```
```

### F.3. State-to-statement conversion prompt

```
You are an expert Lean4 formalization assistant.
Task:
Given an original Lean4 theorem statement for reference, and a tactic state created after
applying some augmentations to the original theorem, create a theorem statement
    corresponding
to the provided augmented state.
Requirements:
1) Keep theorem name and binder structure aligned with the original statement unless the
    augmented state requires changes.
2) Preserve hypothesis order from the augmented state.
3) Preserve the augmented state's mathematics exactly (do not simplify away changes).
4) Replace shorthand notation with explicit Lean forms when appropriate (e.g. `\sqrt2` ->
    `Real.sqrt 2`, `\pi` -> `Real.pi`).
5) Output must contain exactly one block in Lean markdown fence format:
   ```lean4
   theorem ... := by
   ```
6) Do not output any explanation or extra text outside that block.
7) Do not include a proof body, tactics, or sorry; stop exactly at `:= by`.
Original statement:
{original_statement}
Original state:
{original_state}
Augmented state:
{augmented_state}
```
```