WHAT SHOULD A THEOREM PROVER SEE? NAMES, NOTATION, AND STRUCTURE IN LEAN

Anonymous authors
Paper under double-blind review

ABSTRACT

Research on machine theorem proving has mostly improved models and search, and has paid less attention to a more basic choice: how a proof state is represented to the learner. The same proof state can be written in quite different surface forms, which may change the statistical structure a model can exploit. We study this problem in Lean, where a proof state can be written in several equivalent ways and every generated proof can be checked mechanically. We define a representation by three independent choices: *identifiers* (I), preserving or standardizing local variable and hypothesis names; *core syntax* (C), writing terms in conventional notation or in Lean's elaborated core syntax; and *relations* (R), supplying or withholding explicit links between related parts of the state. We introduce ICR, a framework for controlled factorial experiments over these three factors. Training small encoder-decoder models from scratch on Mathlib proof states, with architecture, optimization, and examples held fixed, we compare all eight combinations of I, C, and R. We find that the representation has a large effect on theorem proving, and that the three factors behave very differently. For C, contrary to the common intuition that a more explicit input helps the learner, conventional mathematical notation consistently outperforms Lean's elaborated core syntax: it improves prediction of the next step and proves 24 to 43% more unseen Mathlib theorems, a larger gain than nearly tripling the model size. This advantage persists even when both representations fit within the model context, so input length alone cannot explain it. For I, we find that models use statistical signal in identifier names: renaming variables and hypotheses more than halves, on average, the probability assigned to the author's next proof step, even when that step does not mention them. Standardizing names removes this sensitivity, although its effect on proving theorems varies across benchmarks. For R, removing the explicit links from trained models barely changes the probability of the correct step, suggesting that providing structure does not mean that a model will use it. In our setting, the choice of representation mattered as much as a substantial increase in model scale, and sometimes more.

1 INTRODUCTION

Learned provers now construct proofs that proof assistants can check, and the infrastructure around formal mathematics is maturing with them: Palomar, announced by Tao (2026), registers Lean formalizations together with checks of their proofs and claimed statements. Progress on provers has come from richer training signals, lemma retrieval, and learning to split hard proofs into smaller steps (Han et al., 2022; Yang et al., 2023; Ren et al., 2025). Each of these systems also makes a choice that is rarely tested on its own: *how should a proof state be represented to the learner?*

Lean offers several faithful answers. Take a local hypothesis named `hpos`. The name hints at how its author meant to use it, perhaps that it states a positivity fact, yet renaming it `l3` leaves the mathematics untouched. The surface expression $x + y$ is short and familiar; its elaborated form spells out the operation, the types, and the implicit arguments that Lean inferred. Beyond the text itself, a model could be handed explicit links, from an expression to its type or from a variable occurrence to its binder. None of these choices changes the theorem, but each changes what the model has to learn: a prover never sees the mathematics, only its representation. A more explicit representation

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

may expose useful structure, but it lengthens the input and may add information the model ignores, while incidental conventions such as names and familiar notation may carry regularities inherited from proofs written by humans. Gains attributed to architecture or scale may therefore rest partly on representation choices that are usually fixed rather than studied.

Earlier work gives good reason to expect these choices to matter. Identifier encodings improve proof synthesis in Coq (Sanchez Stern et al., 2023), graph representations affect theorem proving in HOL Light (Paliwal et al., 2020), and structured kernel representations support tactic prediction and the use of new definitions (Yang & Deng, 2019; Blaauwbroek et al., 2024). These studies, however, typically introduce a representation together with other changes to the prover, so it is hard to tell which aspect matters, how the choices interact, and whether the information supplied is used after training. We introduce ICR, a framework that varies names, notation, and structure independently while everything else about learning stays fixed. Our contributions are:

- **The ICR framework.** We formalize the representation of a proof state as a renderer paired with a reversible target codec, and define three independently switchable factors: canonical identifiers (I), elaborated core syntax (C), and typed relations as attention biases (R). A matched protocol fixes the examples, tokenizer, initialization, batches, and updates across all eight cells, and Lean checks every predicted tactic. To our knowledge, ICR is the first framework to vary names, notation, and structure independently for a theorem prover under such controls.
- **A complete factorial study.** We train all eight cells from scratch, so no pretraining corpus favors one notation, at two model sizes and two seeds (32 runs), and evaluate them by tactic likelihood and by verified proof search on unseen Mathlib theorems, miniF2F, and ProofNet Verified.
- **Interventions on trained models.** A renaming of Lean states that preserves their meaning, which Lean confirms reaches the same successor, and the removal or relocation of relation edges test whether a model uses the information it is given.
- **A likelihood accounting identity.** We split the change in loss per token into a loss term and a denominator term, and show that when a representation lengthens the targets, token loss can fall while whole actions become less likely.

Familiar notation yields a better and cheaper prover. Models trained on surface notation prove 24 to 43% more unseen Mathlib theorems than models trained on core syntax, from inputs about one eighth as long, and the advantage persists on states and theorem roots that fit every view. With a quarter of the search budget, every Small run with surface notation proves more Mathlib theorems than every Medium run with core syntax at the full budget, though Medium has 2.7 times the parameters. In this setting, choosing the representation buys more than scaling the model, with far fewer encoder tokens and far less search.

Models exploit the authors' names. Renaming local variables and hypotheses lowers the probability of the author's next tactic by a factor of 2.0 to 2.5 on average, even when the tactic never mentions them. Canonical names remove this dependence exactly, but they do not improve search and they cost miniF2F proofs.

Trained models barely use explicit structure. Deleting the relations from trained models changes tactic likelihood by about 10^{-5} nats per token, even where training with them lowered the loss.

2 Related work

Learning from proof states. CoqGym and ASTactic combine a dataset of interactive proofs with structural term encoders and tactic generation guided by a grammar (Yang & Deng, 2019). PACT derives auxiliary tasks, including elaboration and type prediction, from kernel proof artifacts (Han et al., 2022). LeanDojo provides extraction, interaction, and premise retrieval for Lean (Yang et al., 2023), and DeepSeek-Prover-V2 combines recursive subgoal decomposition, synthetic proofs, and reinforcement learning (Ren et al., 2025). These systems improve the learner or its data; ICR holds both fixed and changes only how each state is presented. Our C factor gives the model elaborated expressions as input, whereas PACT uses them as extra supervision.

Names and structure. Passport enriches identifier encodings with category information, subwords, and qualified paths (Sanchez Stern et al., 2023). Paliwal et al. (2020) compare graph constructions

for HOL Light, including variants that hide variable names, and Graph2Tac represents kernel terms as graphs with explicit binding and learns embeddings for new definitions (Blaauwbroek et al., 2024). Rather than proposing one enriched representation, ICR varies names, syntax, and relations in one factorial design and tests whether trained models use what each supplies.

Relations in attention. Relative position representations (Shaw et al., 2018) and Graphormer (Ying et al., 2021) add relational information to attention logits. Our R factor adds learned scalar biases for typed Lean relations to a standard encoder-decoder Transformer (Vaswani et al., 2017), restricted to directly linked token pairs, with no path distances, centrality terms, or message passing.

3 The ICR framework

3.1 Representations as renderers and codecs

We formalize a representation of a proof state as a pair (ρ, τ). The renderer ρ maps an authoritative Lean state S to an encoder input $\rho(S) = (x, E)$, a token sequence x together with a set E of typed relations between token positions. The target codec τ maps a source tactic a to the sequence the model learns to produce, and its inverse τ_S^{-1} turns a predicted sequence back into a tactic that Lean can execute in S. ICR is the family of eight representations (ρ_{icr}, τ_i), indexed by three bits, with three properties. *Input isolation:* ρ_{icr} reads only S, never the action, the successor, or any outcome. *Factor separation:* i changes the spelling of incidental local identifiers in x and in the target, c changes the syntax of expressions in x, and r changes only E, so the codec depends on i alone. *Executability:* τ_S^{-1} restores the live local names, so Lean can check every generated tactic in every cell. Cell 000 keeps surface notation, the authors' names, and no relations; cell 111 enables all three factors.

I: canonical identifiers. With $i = 1$, the renderer gives local declarations and other incidental identifiers deterministic names that respect thier scopes, and τ_1 rewrites references to existing locals in the target to match. Before Lean runs a prediction, τ_S^{-1} restores the live names. Names that a tactic introduces keep their ordinary Lean scope, and global mathematical names are the same in every cell. Because I changes the target, likelihood comparisons need care (Section 4.2).

C: elaborated core syntax. With $c = 1$, each surface expression is replaced by a compact serialization of Lean's expression constructors, including applications, binders, constants, and universe levels. The model sees elaborated types and implicit arguments, and no definition is unfolded. The grammar is compact, but its output is long: a core state has about eight times as many tokens as its surface form on average. Appendix A gives the grammar and one test state in all four textual views.

R: typed attention biases. With $r = 1$, E contains edges of seven relation types, such as binder references, types, and application arguments (Appendix Table 3); argument edges also activate one of eight position buckets. Exact character offsets project each endpoint to a single token. At encoder layer ℓ and head h, the attention logit is

$$A_{uv}^{\ell h} = \frac{q_u^{\ell h} \cdot k_v^{\ell h}}{\sqrt{d_h}} + r \sum_{f=1}^{15} b_f^{\ell h} e_f(u, v), \tag{1}$$

where $e_f(u, v)$ indicates that feature f is active on the directed pair (u, v). The seven types and eight buckets give the 15 features, each counted once per pair, and only encoder self attention is biased. Edges whose endpoints do not appear in the text are dropped and counted: core text supplies both endpoints for every edge, but surface text does so for only 11.02% of them (Section 4.4). Every cell has the same bias parameters, initialized to zero and with the same weight decay, and $r = 0$ supplies no edges, so tokens, targets, and parameter counts do not depend on R.

3.2 One population and a matched protocol

Each example pairs a tactic a_t written in the Mathlib source with the Lean state S_t immediately before it. With Lean and Mathlib pinned at version 4.32.1, we keep the 248,698 Mathlib transitions that replay to their recorded successors. Of these, 237,159 complete our deterministic rendering within

fixed resource bounds, and this population is shared by every cell. We split it 90/5/5 by proposition cluster, grouping theorems whose statements match after normalizing incidental names, which gives 213,439 training, 11,873 validation, and 11,847 test transitions. A single byte level BPE tokenizer with 16,384 entries is fitted to the training split, pooling the four textual views and the two target forms. A transition is kept only if every view fits in 4,096 tokens and both target forms fit in 512, so every model sees the same 137,880 training, 7,633 validation, and 7,616 test examples.

Small and Medium models have 11.54M and 31.08M parameters, with four or six layers per stack, widths of 256 or 384, and four or six attention heads. Within each size and seed, all eight cells start from the same parameters and see the same examples in the same order, with identical batch membership and padding shapes set by the longest view. Every run takes 12,927 updates over three epochs at an effective batch size of 32, and we evaluate the final checkpoint. Appendix A gives the architecture, optimizer, and compute.

3.3 Verified search and effects

Our primary measure is Solve@64, the fraction of theorems proved within 64 search expansions, with every proof checked by Lean. The internal benchmark consists of 512 clusters drawn deterministically from the 4,901 test theorem clusters, before the length filter. At each expansion, beam search proposes eight tactic candidates of at most 64 tokens, including the end token, ranked by tactic log probability. A priority queue orders each state s_d by the log probability of the tactic path $a_1, \ldots, a_d$ that first reached it from the root s_0,

$$\mathrm{score}(s_d) = \sum_{k=1}^{d} \log p_\theta(a_k \mid s_{k-1}), \tag{2}$$

without length normalization. Lean runs each candidate under a fixed heartbeat budget; failed candidates and unchanged or repeated states are discarded, and the theorem being proved and all later declarations are hidden from the environment. Solve@16 and Solve@32 are read from prefixes of the same trace. We also evaluate on all 244 miniF2F test problems (Zheng et al., 2022) and all 367 records of the pinned ProofNet Verified release (Azerbayev et al., 2023; ProofNet Verified contributors, 2026) in the same environment, giving the model only the formal state. Every accepted proof is recompiled and checked by the kernel, with axioms limited to `propext`, `Classical.choice`, and `Quot.sound`, and execution errors and overlength states count as failures.

We report two likelihood measures on the 7,616 test examples: token NLL, the summed target loss divided by the number of target tokens, and complete action NLL, the same sum divided by the number of actions. For search, let $Y_j(i, c, r)$ indicate whether theorem j is solved in a given size and seed block. The effect of I is the paired difference averaged over the other two factors,

$$\widehat{\Delta}_I = \frac{100}{4n} \sum_{j=1}^{n} \sum_{c,r \in \{0,1\}} \big[Y_j(1, c, r) - Y_j(0, c, r) \big], \tag{3}$$

and the effects of C and R are defined in the same way. We keep blocks separate and report 95% intervals from 2,000 paired bootstrap draws, resampling proposition clusters for losses and theorem records for search. When we give one number per block, the order is Small seed 0, Small seed 1, Medium seed 0, Medium seed 1.

4 Empirical results

4.1 Familiar notation beats elaborated syntax

Core syntax gives the model information that surface notation leaves implicit, including inferred types, implicit arguments, and type class instances, yet it yields a weaker learner. Models trained on core inputs have 0.080 to 0.095 nats higher token NLL and 1.644 to 1.957 nats higher complete action NLL across the four blocks, averaging over I and R. Because C leaves the target unchanged, these comparisons are direct. The gap in training loss opens within about 2,000 updates and persists (Appendix B).

The gap is larger still in proof search (Table 1 and Figure 1). The average C effect on Mathlib Solve@64 ranges from -2.64 to -4.74 percentage points, and summed over the four cells in each

Table 1: Theorems proved within 64 expansions for all 32 factorial runs, given as seed 0 / seed 1. Small and Medium are the two model sizes, with 11.54M and 31.08M parameters. Cell bits denote I, C, and R, with 1 meaning the factor is enabled. Every cell has the same denominators.

	Small			Medium		
Cell	Mathlib /512	miniF2F /244	ProofNet /367	Mathlib /512	miniF2F /244	ProofNet /367
000	69 / 71	51 / 59	16 / 19	78 / 83	63 / 69	18 / 19
001	72 / 71	52 / 58	16 / 19	75 / 85	62 / 68	18 / 19
010	60 / 57	40 / 41	20 / 19	58 / 59	47 / 57	17 / 18
011	58 / 53	39 / 40	18 / 19	59 / 58	49 / 67	17 / 18
100	73 / 69	47 / 39	20 / 21	72 / 77	51 / 59	21 / 22
101	74 / 69	51 / 41	21 / 20	72 / 77	50 / 61	20 / 22
110	54 / 57	49 / 42	19 / 19	61 / 55	47 / 49	20 / 20
111	54 / 59	47 / 43	19 / 18	59 / 53	46 / 51	19 / 22

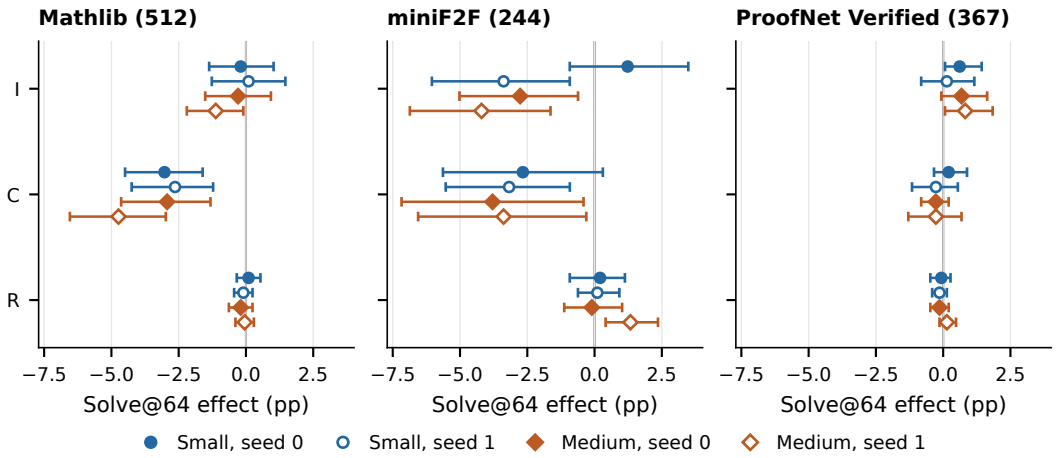

Figure 1: Factorial effects on verified theorem completion. Each point is the paired effect of enabling one factor, averaged over the other two; positive values favor enabling it. Circles: Small; diamonds: Medium; filled: seed 0; open: seed 1. Bars show 95% paired bootstrap intervals.

block, surface runs prove 24 to 43% more theorems than core runs. Surface models also solve more miniF2F problems in all four blocks. On ProofNet Verified we find no consistent C effect: the four averaged effects range from -0.27 to $+0.20$ points, and every interval includes zero. Surface models need far less search, too. In all 16 matched pairs of cells, the surface model proves more Mathlib theorems within 16 expansions than its core counterpart within 64, and every Small surface run at 16 expansions (66 to 72 theorems) beats every Medium core run at 64 (53 to 61; Appendix Table 8); summed over each block, the same holds on miniF2F. For comparison, moving from Small to Medium, with 2.7 times the parameters, adds only 0.49 points at seed 0 and 1.00 point at seed 1, averaged over cells.

An obvious explanation is input length. Core inputs average 1,589.3 tokens against 186.8 for surface inputs with original names, and 1,668.3 against 209.8 with canonical names, ratios of 8.51 and 7.95. Three checks show that the gap is not an artifact of the context limit (Figure 2).

First, the likelihood gap appears on short states. In the shortest stratum, 689 test states whose four views all fit in 512 tokens, one eighth of the encoder limit, core models still incur 0.086 to 0.107 nats more loss per token. The penalty is positive in every longer stratum and in every quartile of the ratio between core and surface length.

Second, the search gap survives when we keep only theorems whose root fits. Of the 512 Mathlib roots, 49 exceed the context limit in original core syntax and 52 in canonical core syntax, while every surface root fits. On the 460 roots that fit all views, the C effects are -2.28, -1.74, -2.01, and

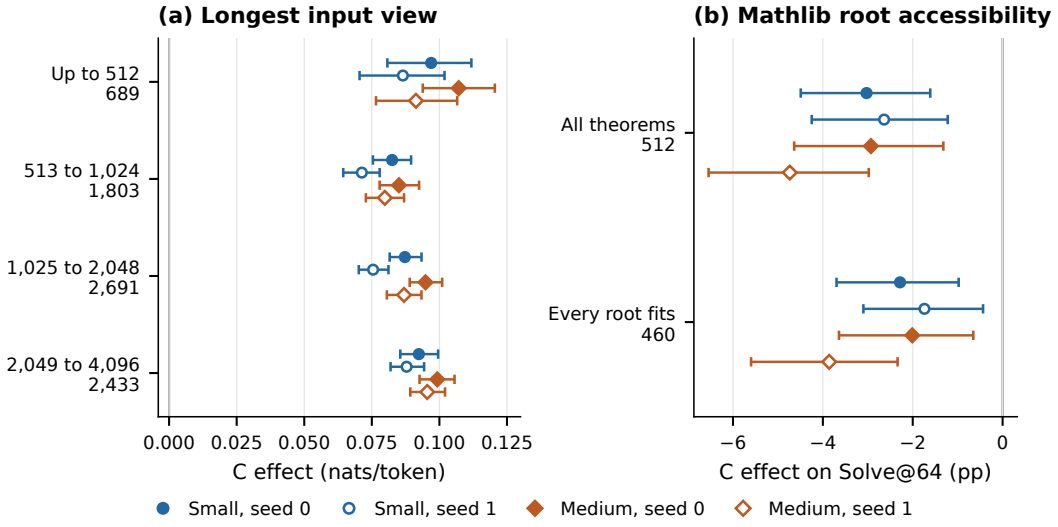

Figure 2: The advantage of surface syntax persists away from the context limit. (a) Paired C effects on token NLL within strata defined by the longest of the four input views, with counts beneath the ranges; positive values mean higher loss with core syntax. (b) The C effect on Mathlib proof completion among the 460 theorems whose root fits every view. Both subsets are defined from inputs alone. Bars show 95% paired bootstrap intervals.

-3.86 points. On miniF2F, where only two core roots are too long, surface models solve more of the 241 roots observed to fit every view in all four blocks.

Third, the gap appears before length can interfere. A root that fits does not guarantee successors that fit: among the 460 theorems, 9.95 to 12.45% of core searches select an intermediate state that exceeds the limit and must be discarded, and no surface search ever does (Table 10). The first eight candidates at the root, however, are generated before any successor is encoded, and completions from these candidates alone still favor surface syntax, with effects of -0.76, -0.49, -1.68, and -1.74 points and intervals that exclude zero for both Medium blocks.

One plausible explanation is the match between input and output. Tactics are written in surface syntax, so surface inputs share their notation, whereas core inputs keep the fully qualified constants and the local names but drop the notation. Elaborated terms can be useful as auxiliary supervision (Han et al., 2022); presented directly as the proof state, they gave a weaker prover here.

4.2 Models exploit the authors' names

With surface inputs, enabling I lowers token NLL by 0.095 to 0.126 nats, averaging over R in each block. Taken at face value, canonical names make tactics easier to predict, but I also changes the tactics themselves: rewriting local references raises the number of test target tokens from 153,527 to 161,400, an increase of 5.13%, so the model is scored on more tokens for the same set of actions.

Write U_i for the summed action NLL with I set to i, T_i for the number of target tokens, and N for the number of actions. Token NLL is U_i/T_i and complete action NLL is U_i/N, and the change in token NLL splits exactly into a loss change at the original denominator and the effect of changing the denominator:

$$\frac{U_1}{T_1} - \frac{U_0}{T_0} = \frac{U_1 - U_0}{T_0} + U_1\left(\frac{1}{T_1} - \frac{1}{T_0}\right). \tag{4}$$

For Medium seed 1 with surface inputs, the two terms are $+0.01256$ and -0.10766 nats per token, a net change of -0.09510. In this block the total loss actually rises, by 0.253 nats per action; token NLL falls only because a larger total is divided by a still larger token count. No other block shows this: in the other three, complete action NLL falls by 0.160, 0.244, and 0.003 nats per action (Table 7), yet the denominator term still accounts for at least 90% of the reduction in token NLL in every block.

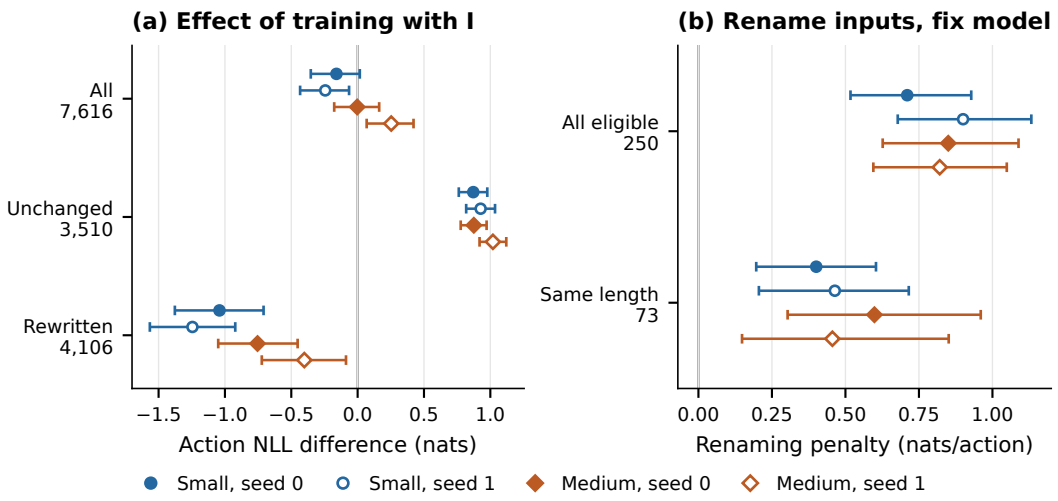

Figure 3: Two views of identifier canonicalization. (a) Training with I moves complete action NLL in opposite directions for targets that I rewrites and for targets it leaves unchanged (surface inputs, averaged over R; lower is better). (b) Renaming locals raises the NLL of unchanged gold actions in fixed 000 models, even on the 73 states whose token counts do not change; canonical 100 models are exactly invariant. Bars show 95% paired cluster bootstrap intervals; markers follow Figure 1.

Splitting the test set by target sharpens the picture (Figure 3a). I rewrites the target bytes of 4,106 of the 7,616 test examples, and across syntaxes and blocks, models trained with I have 0.402 to 1.825 nats lower complete action loss on rewritten targets but 0.494 to 1.019 nats higher loss on the 3,510 unchanged ones. Because the two groups contain different actions, this split cannot isolate the names.

Renaming with the model and target fixed. To isolate the names, we intervene on trained models. We deterministically select test states whose target bytes are unchanged under I and replace eligible local names with fresh aliases inside a copied Lean state. The 250 eligible examples keep declaration identity, types, binding structure, and the executable gold tactic, and Lean confirms that the gold tactic reaches the same semantic successor from both versions. We then score the saved 000 and 100 models from all four blocks on both versions.

Renaming costs the models with original names 0.710, 0.900, 0.850, and 0.821 nats per action, and every interval lies above zero. Put differently, the geometric mean probability of the gold tactic falls by a factor of 2.03 to 2.46. On the 73 examples where renaming leaves the input token count unchanged, the penalty is still 0.401 to 0.599 nats.

The canonical models see the same input before and after renaming. If g is such a renaming, the canonical renderer ρ_{100} maps S and gS to the same encoder input, so for fixed parameters θ

$$\rho_{100}(gS) = \rho_{100}(S) \quad \Longrightarrow \quad p_\theta(y \mid \rho_{100}(gS)) = p_\theta(y \mid \rho_{100}(S)), \tag{5}$$

simply because a deterministic model receives identical inputs. We confirmed this on all 250 states: canonical tokens and action NLLs are unchanged, and so are the beam outputs on the generation subset. Even after renaming, the canonical models assign lower probability to the gold tactics than the models with original names do; the gap narrows from 0.945 to 1.220 nats per action before renaming to 0.044 to 0.399 after it. Canonicalization thus removes the sensitivity to renaming, but also discards information carried by the names.

4.3 Canonical names improve invariance, not search

Invariance to renaming does not turn into more proofs (Figure 1). With surface inputs, I changes miniF2F Solve@64 by -1.02, -7.58, -4.92, and -3.48 points, and the last three intervals exclude zero. On ProofNet Verified the effects are small, 0.41 to 1.23 points or 1.5 to 4.5 theorems per cell, and only the Small seed 0 interval excludes zero. On Mathlib the surface I effects run from -1.37

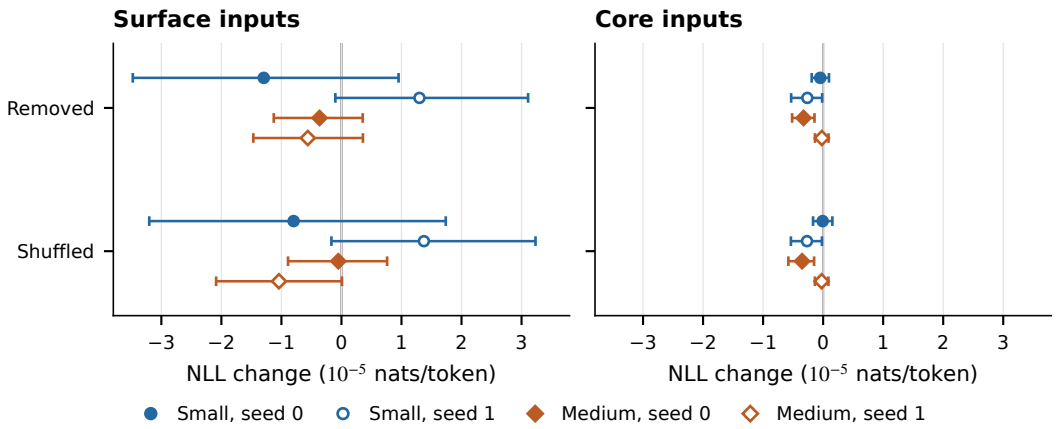

Figure 4: Direct use of relations in eight fixed $R = 1$, $I = 0$ checkpoints. Removing all relation edges, or consistently permuting their token endpoints (which keeps labels and topology), barely changes the average NLL of the gold tactic on 1,024 states. Units are 10^{-5} nats per token; ordinary NLL is about 2.2 to 2.5 nats per token. Bars show 95% paired cluster bootstrap intervals.

to $+0.59$ points, and every interval includes zero. Root diagnostics show no consistent mechanism (Appendix B).

The search procedure could also explain this. The frontier score in Equation 2 is not normalized by length, and I lengthens tactics that name local hypotheses, so canonical paths might be ranked too low. Two checks argue against this. Across both R settings and all four blocks, 128 miniF2F problems are solved with original names but not with canonical names. In 114 of them, the canonical search exhausted its frontier within the budget, having expanded every reachable state, so no other ordering could have found a proof. We also reran miniF2F for cells 000 and 100 with the frontier ordered by log probability per token and the beam unchanged. At most four outcomes change in any run, and no 100 minus 000 difference moves by more than 0.82 points (Appendix B.6).

The theorems gained with canonical names also differ between seeds: of the 12 and 11 Mathlib solves that Small 100 adds over Small 000 at the two seeds, only two recur (Appendix E).

4.4 Relation biases are received but barely used

The links that R adds do reach the model. Across the test population, 11.02% of semantic edges have both endpoints in the original surface view, all of them do in the core view, and every evaluated state has projected pairs. The learned biases are nonzero, with mean absolute values from 0.0100 to 0.0370 and maxima from 0.0929 to 0.3032 across the eight checkpoints used below.

To test whether the models rely on them, we fix eight checkpoints with $I = 0$ and $R = 1$, covering both sizes, both seeds, and both syntaxes, and score 1,024 test states from 899 proposition clusters in three ways: with the real relations, with no relations, and with relations whose token endpoints are consistently permuted. Tokens, targets, weights, and padding stay the same. Average likelihood barely moves: across all 16 removal and shuffle contrasts, the mean change lies between -1.29×10^{-5} and 1.37×10^{-5} nats per token, and every 95% interval lies within $\pm 3.5 \times 10^{-5}$ (Figure 4).

For Small seed 0 with core inputs, the model trained with relations has lower loss on these 1,024 states than its counterpart trained without them: the R effect is -0.00342 nats per token, with interval $[-0.00561, -0.00121]$. Removing the edges from that checkpoint, however, changes its loss by only -0.048×10^{-5} nats per token (Table 14), so it keeps essentially all of its advantage without them.

Because surface text carries only about a ninth of the the edges, we also compare the trained R effect between syntaxes. With core inputs, R lowers token NLL more than with surface inputs in all four blocks, but the difference is small, 0.18×10^{-3} to 0.97×10^{-3} nats per token, or about one percent of the C effect at most, and its intervals exclude zero in two blocks. For Solve@64, the difference

between the core and surface R effects includes zero in 11 of the 12 combinations of benchmark and block (Table 11).

5 Conclusion

We introduced ICR, a framework that turns what a theorem prover should see into a controlled measurement, and used it to train 32 provers, at two sizes and two seeds, that within each size and seed differed only in whether each Mathlib proof state kept its authors' identifiers (I), was written in surface notation or core syntax (C), and came with explicit relations (R). We showed that the conventions of human mathematics carried signal that models used, while a more explicit or more structured representation was not by itself more useful. Surface notation gave 24 to 43% more verified Mathlib proofs than core syntax from inputs one eighth as long, and with a quarter of the search budget every Small surface run outproved every Medium core run at full budget. Models trained with the authors' names relied on them: renaming alone more than halved the probability of the author's next step, a dependence that evaluations with the original names cannot reveal, and canonical names traded this signal for exact invariance at the price of some miniF2F proofs. We also showed that when a representation lengthens the targets, token loss can improve while whole actions become less likely, so representations are better compared by likelihood per action and by verified search. Relation biases lowered one model's loss, yet deleting them afterward changed almost nothing, so supplying structure did not mean that the model used it. Future work should test whether these effects hold at more model sizes and seeds and for larger, pretrained provers, and should separate the parts of the penalty for core syntax: longer inputs, explicit types and implicit arguments, qualified names, and missing notation. Richer structural encoders, such as message passing, may also make use of the relations that scalar attention biases leave unused. The representation of a proof state should therefore be evaluated as carefully as the model and the search procedure.

Limitations

The original plan had three model sizes and three seeds (72 runs); compute and time limits reduced it to the 32 runs reported here. The second seed was added after results for the first were available, so it is best read as a replication, not part of a preregistered design. The external benchmarks, the interventions, and the four PAIR runs, which add structural annotations to cell 101 (Appendix D), are exploratory. All models are small and trained from scratch, and the effects may weaken or reverse for larger or pretrained provers. The two sizes do not share initial parameters, so comparisons between them are unpaired; our intervals hold the trained models fixed, so training variation appears only across seeds. Our checks show that the penalty for core syntax is not an artifact of the context limit, but they do not separate longer sequences, explicit types and implicit arguments, qualified names, and missing notation. The relation results concern one mechanism, scalar biases on directly linked token pairs, which touch 0.205% of surface and 0.0328% of core token pairs; richer graph encoders or message passing may use the same information better. We did not compare against pretrained step provers such as ReProver (Yang et al., 2023) or tune search scoring beyond one check with length normalization. The interventions use teacher forcing on subsets of states and checkpoints, not search, and small average changes can hide larger individual ones. Two ProofNet propositions also occur exactly in our Mathlib split, but no model solves either.

Reproducibility statement

Section 3 and Appendix A describe the population, split, tokenizer, models, optimizer, and search protocol. The code for extracting and rendering the Mathlib population, training the models, and running verified search is available at `https://anonymous.4open.science/r/iclr-27-FB07/README.md`.

AI use statement

An LLM edited the wording of the abstract, introduction, Section 4, and conclusion, adjusted the layout, and helped prepare the code release; we reviewed all of it and take full responsibility.

REFERENCES

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. ProofNet: Autoformalizing and formally proving undergraduate level mathematics. arXiv:2302.12433, 2023. URL https://arxiv.org/abs/2302.12433.

Lasse Blaauwbroek, Miroslav Olšák, Jason Rute, Fidel Ivan Schaposnik Massolo, Jelle Piepenbrock, and Vasily Pestun. Graph2Tac: Online representation learning of formal math concepts. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2401.02949.

Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, 2021. URL https://lean-lang.org/papers/lean4.pdf.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. Proof artifact co training for theorem proving with language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2102.06203.

Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher order logic and theorem proving. In *AAAI Conference on Artificial Intelligence*, 2020. URL https://doi.org/10.1609/aaai.v34i03.5689.

ProofNet Verified contributors. ProofNet Verified: Audited and corrected lean 4 benchmark. Dataset repository, 2026. URL https://github.com/marcusm117/ProofNet-Verified/tree/160414332dc196583f6c37c310b420d2a3b07c58. Revision 160414332dc1.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. DeepSeek Prover V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. arXiv:2504.21801, 2025. URL https://arxiv.org/abs/2504.21801.

Alex Sanchez Stern, Emily First, Timothy Zhou, Zhanna Kaufman, Yuriy Brun, and Talia Ringer. Passport: Improving automated formal verification using identifiers. *ACM Transactions on Programming Languages and Systems*, 45(2), 2023. URL https://doi.org/10.1145/3593374.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self attention with relative position representations. In *North American Chapter of the Association for Computational Linguistics*, 2018. URL https://aclanthology.org/N18-2074/.

Terence Tao. Palomar: A registry of Lean verified mathematics. What's New, 18 August, 2026. URL https://terrytao.wordpress.com/2026/08/18/palomar-a-registry-of-lean-verified-mathematics/.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. URL https://arxiv.org/abs/1706.03762.

Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning*, 2019. URL https://arxiv.org/abs/1905.09381.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval augmented language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2306.15626.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie Yan Liu. Do transformers really perform bad for graph representation? In *Advances in Neural Information Processing Systems*, 2021. URL https://arxiv.org/abs/2106.05234.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. miniF2F: A cross system benchmark for formal olympiad level mathematics. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2109.00110.

A EXPERIMENTAL PROTOCOL

A.1 DATA AND REPRESENTATION

Lean separates source syntax, elaborated terms, tactic execution, and kernel proof checking (de Moura & Ullrich, 2021). We use this separation to vary the presentation of a fixed before state. Replay certification checks the source action against the saved state and its recorded successor. Materialization then renders each retained before state in the four distinct textual views. The relation switch supplies a separate side channel, giving eight conditions from those four texts. Every record retains its original transition identity throughout this process.

The split uses exact canonical proposition identity after normalizing incidental names and metadata. Proposition clusters stay together, with assignment seed 1729. The length rule is applied after splitting and requires simultaneous support under all representations. Table 2 gives the resulting populations. The 7,616 model test transitions belong to 3,793 proposition clusters. The internal search sample is drawn from the full set of 4,901 test clusters before the model length rule is applied.

Table 2: Shared transition populations. Materialization and the model length rule are applied once for all eight conditions.

Population	Training	Validation	Test	Total
Replay certified				248,698
Materialized and split	213,439	11,873	11,847	237,159
Common model support	137,880	7,633	7,616	153,129

The tokenizer uses byte level BPE, with 16,384 vocabulary entries and reserved atomic state markers. It is fitted only to the training partition, pooling the four input texts and two target forms. It applies neither Unicode normalization nor tokenizer dropout. The encoder stores beginning and end tokens; the decoder predicts target tokens and the end token. Padding and the decoder's initial token do not contribute to NLL. The common length limits are inclusive of the stored boundary tokens, and admitted sequences are retained in full.

The canonicalizer gives deterministic scoped names to local declarations, bound variables, goal labels, and metavariables. The action codec rewrites references to existing locals while preserving the scope of identifiers introduced within a tactic. Decoding consults the current state's inverse map before executing a candidate. Core syntax serializes Lean expression constructors and universe information without unfolding definitions. Both C and R preserve the action target.

Core syntax grammar. Before serialization, assigned metavariables are instantiated and metadata nodes are removed; no definition is unfolded. Every expression e and universe level u is then written as

```
e ::= bvar(x;i) | fvar(x) | mvar(m) | sort(u) | app(e,e)
    | const("c";[u,...,u]) | lam(x;b;e;e) | forall(x;b;e;e)
    | let(x;d;e;e;e) | nat(n) | string("s") | proj("S";i;e)
u ::= 0 | succ(u) | max(u,u) | imax(u,u) | "p" | ?v
```

Here `i` is a de Bruijn or projection index, `b` is one of `default`, `implicit`, `strictImplicit`, and `instImplicit`, `d` is `dep` or `nondep`, `"p"` is a universe parameter, and `?v` is a universe metavariable. Every application is binary, so a function applied to k arguments, including implicit and instance arguments, nests k `app` nodes. Global constants are always fully qualified. Local, bound, and metavariable names follow the I setting.

Example. The following test state comes from `Nat.right_le_pair`. We show one hypothesis and the target in each textual view; the full state has 77, 88, 219, and 230 tokens in cells 000, 100, 010, and 110.

Surface, original names: `h : ¬a < b`, with target `b` ≤ `Nat.pair a b`
Surface, canonical names: `l3 : ¬l1 < l2`, with target `l2` ≤ `Nat.pair l1 l2`

Core, original names:

```
h :  app(const("Not";[]),app(app(app(app(
    const("LT.lt";[0]),const("Nat";[])),
    const("instLTNat";[]))),fvar(a)),fvar(b)))
target:  app(app(app(app(const("LE.le";[0]),
    const("Nat";[]))),const("instLENat";[])),
    fvar(b)),app(app(const("Nat.pair";[]),
    fvar(a)),fvar(b)))
```

Core, canonical names: the same, with `l1`, `l2`, and `l3` in place of `a`, `b`, and `h`.

Gold tactic: `· simp [pair, h]`; under I, `· simp [pair, l3]`.

Core syntax makes the instances `instLTNat` and `instLENat`, the type argument `Nat`, and the universe levels explicit, which surface notation leaves implicit.

A.2 Relation projection

Table 3 lists the seven semantic relation types. An application argument edge activates its base type and one position feature, with buckets $0, 1, \ldots, 6, \geq 7$. This gives 15 learned scalars for each encoder layer and head. Relation features on the same directed token pair add according to Equation 1; repeated instances of the same feature contribute once. The biases enter only encoder self attention. Each layer stores its scalars as a matrix over heads and features.

Table 3: Semantic information exposed by R. Feature names identify the frozen relation schema.

Feature	Relationship
`BINDER_REF`	Bound occurrence and its binder
`TYPE_OF`	Expression and its type
`APP_ARG`	Application and an argument, with its position
`LOCAL_DEPENDS_ON`	Local declaration and a local dependency
`GLOBAL_REF`	Constant occurrence and its global reference
`GOAL_TARGET`	Goal and its target expression
`SHARED_METAVAR`	Occurrences sharing a metavariable

Endpoints are aligned by exact serialized character spans and mapped to the first overlapping tokenizer position. The edge source indexes the query and its destination indexes the key. Entity lookup prefers a serialized declaration when present and uses the corresponding reference alignment otherwise. Projection records unavailable endpoints and drops only the affected edge. Across the model test set, original surface text projects 483,235 of 4,385,417 semantic edges; canonical surface text projects 483,446. Both core views project all 4,385,417. Surface notation hides much of the elaborated structure, making this difference in coverage a measurable property of the representations.

A.3 Model and optimizer

Table 4 gives the two model sizes, both trained from scratch. Each stack uses normalization before its blocks, RMSNorm, rotary position encodings, and GELU feed forward layers. Encoder and decoder embeddings and the output matrix share weights. Dropout is 0.1. Attention and other projections have no additive bias apart from the specified relation terms. All cells include the relation parameters, initialized to zero, so the parameter counts and initial predictions are matched within each size and seed block.

Table 4: Architecture. The parameter totals include relation scalars in every condition.

Size	Enc. layers	Dec. layers	Width	Heads	FF width	Parameters
Small	4	4	256	4	1,024	11,540,208
Medium	6	6	384	6	1,536	31,076,892

The objective is token normalized cross entropy. AdamW uses $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, and weight decay 0.1 on parameters with at least two dimensions, including the relation bias matrices. Peak learning rates are 3×10^{-4} for Small and 2×10^{-4} for Medium. The schedule warms up linearly for 5% of updates and then decays linearly; gradient norm is clipped at 1.0. The schedule and hyperparameters are shared across cells within each size. Training uses BF16 and activation checkpointing. Exact encoder attention is evaluated in chunks of 128 queries, using the same implementation with or without relations.

Length buckets use each row's maximum length across the four textual views. The eight cells share transition order, microbatch membership, accumulation boundaries, and padded shapes. Accumulation gives an effective batch size of 32, with a final partial update at each epoch boundary. Each completed run processes 413,640 examples over three epochs and 12,927 updates. Checkpoints are saved every 250 updates, and evaluation uses the final checkpoint.

Table 5: Measured training time for the 32 factorial runs. Hours sum the elapsed times recorded for optimizer updates on one A100 GPU per run. They exclude preprocessing, validation, checkpoint writing, and evaluation.

Size	Runs	Total GPU hours	Hours per run
Small	16	313.1	15.2 to 30.9
Medium	16	604.8	29.9 to 56.8

All 36 completed models, including PAIR, trained on an NVIDIA A100 with 40 GB memory. The runtime records identify Python 3.11.9 and PyTorch 2.11.0 with CUDA 13.0. Factorial optimizer updates account for 917.9 GPU hours; including the four PAIR runs gives 1,024.1 hours under the same accounting convention. Because every cell in a block is padded to the same shapes, cells without relations take about the same time with surface and core inputs. Relation biases add to the time per update: with surface inputs, $R = 1$ cells took 14 to 19% longer than the matching $R = 0$ cells, and with core inputs 1.8 to 2.0 times as long.

A.4 Search and benchmark identities

An expansion selects one frontier state, generates up to eight completed tactic candidates by deterministic beam search, and submits them to Lean in rank order. A candidate has at most 64 generated tokens including its end token. The beam and frontier use unnormalized log probabilities, including the end token, and deterministic tie rules; Equation 2 gives the frontier score. Model evaluation uses FP32. Lean receives 200,000 heartbeats per candidate. The worker also has a 300 second response guard. Unchanged states and duplicate successors are pruned, and overlength states cannot be expanded. Solve counts at 16 and 32 expansions are prefixes of the same trace used for 64 expansions.

The internal search root is the theorem's legal declaration position. Its environment excludes the theorem being proved and later declarations. The 512 theorem sample is fixed by a deterministic hash selection. The external benchmark sources are the miniF2F Lean 4 repository at revision `5746b7d6c478` and ProofNet Verified at revision `160414332dc1`. The former contributes its 244 test problems; the latter contributes all 367 indexed records, containing 366 distinct declaration names. Each record retains its own benchmark identity.

Both external suites are elaborated under the experiment's Lean and Mathlib 4.32.1 environment. A compatibility alias maps `LocPathConnectedSpace` to `LocallyPathConnectedSpace`; it does not change the proposition. The model receives the formal state without the informal problem or supplied proof. Completed external paths are compiled again as proofs of their original statements, checked by the kernel, and checked for the permitted axiom set. Resource failures retain their original theorem record and contribute zero to the solve indicator.

B ADDITIONAL FACTORIAL RESULTS

B.1 LIKELIHOOD AND TRAINING TRAJECTORIES

Table 6 reports token NLL for all 32 factorial checkpoints. Figure 5 follows the core syntax contrast throughout training. Each point summarizes a window of up to 250 updates, computes each cell's total loss divided by its predicted target tokens, and averages the paired C difference over I and R.

Table 6: Test token NLL in nats on all 7,616 model test transitions. Lower is better. The target token totals differ under I, as analyzed in Section 4.2.

Cell	Small, seed 0	Small, seed 1	Medium, seed 0	Medium, seed 1
000	2.358336	2.356506	2.197743	2.194353
001	2.358120	2.356456	2.197784	2.194692
010	2.460892	2.449230	2.313248	2.301958
011	2.458608	2.449224	2.312850	2.302822
100	2.235957	2.229657	2.090263	2.099409
101	2.235360	2.230382	2.090563	2.099435
110	2.311134	2.297176	2.166057	2.169625
111	2.312112	2.296717	2.164863	2.168767

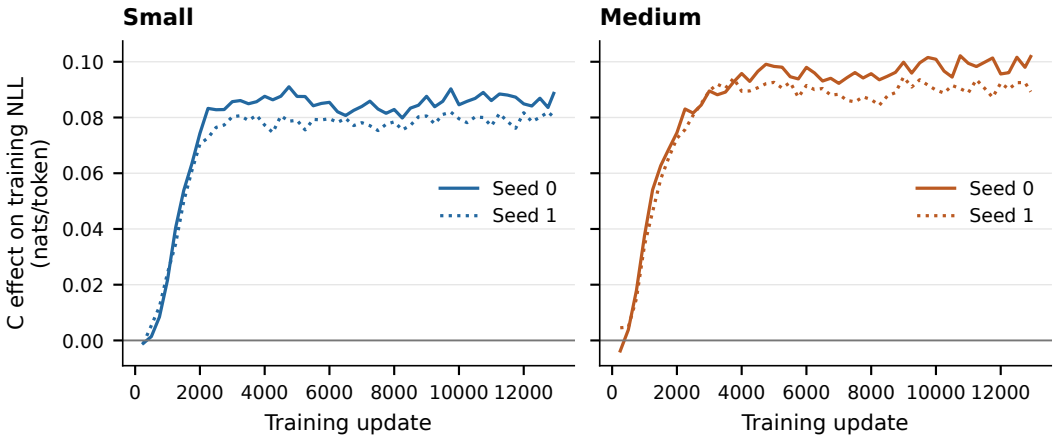

Figure 5: The C effect on training token NLL, averaged over I and R. Positive values mean higher loss with core inputs. The gap develops early and persists in both model sizes and seeds.

We find a larger reduction in complete action NLL from canonicalizing names when the model sees core inputs (Table 7). Subtracting the surface I effect from the core I effect gives a negative difference in every block, with all four paired 95% intervals below zero.

Table 7: Interaction of names and syntax in complete action NLL. Each I contrast averages over R. The final column subtracts the surface contrast from the core contrast, giving a difference of differences in nats per action. Differences are computed before rounding.

Size	Seed	I effect, surface	I effect, core	Difference
Small	0	−0.160	−0.596	−0.437
Small	1	−0.244	−0.695	−0.451
Medium	0	−0.003	−0.737	−0.734
Medium	1	+0.253	−0.443	−0.696

B.2 SOLVE COUNTS AT EVERY SEARCH BUDGET

Table 8 gives solve counts at 16, 32, and 64 expansions for every factorial run. On Mathlib, every surface cell at 16 expansions proves more theorems than its matched core cell at 64, in all four blocks.

Table 8: Theorems proved within 16 / 32 / 64 expansions for all 32 factorial runs. Counts at smaller budgets are prefixes of the same search trace. Cell bits denote I, C, and R.

Cell	Small, seed 0	Small, seed 1	Medium, seed 0	Medium, seed 1
Mathlib, /512				
000	67 / 69 / 69	68 / 68 / 71	76 / 78 / 78	81 / 83 / 83
001	70 / 72 / 72	67 / 68 / 71	73 / 75 / 75	83 / 85 / 85
010	57 / 60 / 60	55 / 57 / 57	57 / 58 / 58	59 / 59 / 59
011	56 / 56 / 58	49 / 53 / 53	59 / 59 / 59	57 / 58 / 58
100	70 / 72 / 73	66 / 67 / 69	67 / 71 / 72	74 / 76 / 77
101	72 / 73 / 74	67 / 68 / 69	68 / 71 / 72	74 / 76 / 77
110	53 / 54 / 54	56 / 56 / 57	57 / 59 / 61	55 / 55 / 55
111	54 / 54 / 54	58 / 58 / 59	59 / 59 / 59	53 / 53 / 53
miniF2F, /244				
000	49 / 51 / 51	58 / 59 / 59	63 / 63 / 63	69 / 69 / 69
001	50 / 52 / 52	57 / 57 / 58	62 / 62 / 62	68 / 68 / 68
010	40 / 40 / 40	41 / 41 / 41	45 / 45 / 47	57 / 57 / 57
011	38 / 39 / 39	40 / 40 / 40	47 / 48 / 49	67 / 67 / 67
100	46 / 46 / 47	38 / 38 / 39	48 / 51 / 51	57 / 59 / 59
101	51 / 51 / 51	40 / 41 / 41	46 / 48 / 50	59 / 61 / 61
110	49 / 49 / 49	41 / 42 / 42	47 / 47 / 47	46 / 49 / 49
111	46 / 47 / 47	41 / 43 / 43	46 / 46 / 46	50 / 50 / 51
ProofNet Verified, /367				
000	16 / 16 / 16	19 / 19 / 19	18 / 18 / 18	19 / 19 / 19
001	16 / 16 / 16	19 / 19 / 19	18 / 18 / 18	19 / 19 / 19
010	20 / 20 / 20	19 / 19 / 19	17 / 17 / 17	18 / 18 / 18
011	18 / 18 / 18	19 / 19 / 19	17 / 17 / 17	18 / 18 / 18
100	20 / 20 / 20	21 / 21 / 21	21 / 21 / 21	22 / 22 / 22
101	21 / 21 / 21	19 / 20 / 20	20 / 20 / 20	22 / 22 / 22
110	19 / 19 / 19	18 / 19 / 19	20 / 20 / 20	20 / 20 / 20
111	19 / 19 / 19	17 / 18 / 18	19 / 19 / 19	22 / 22 / 22

B.3 WHAT A ROOT CANDIDATE ACCOMPLISHES

Table 9 follows the naming effect from execution to completion on miniF2F. For each theorem we ask whether any of the first eight root candidates executes (valid), changes or closes the state (progress), or completes the proof, with a fixed denominator of 244 theorems. At Medium seed 1, canonical names raise the valid and progress rates by 7.79 and 5.74 points but lower root and final completion by 3.69 and 3.48 points, with 95% intervals $[3.89, 11.89]$, $[1.64, 10.25]$, $[-6.97, -0.61]$, and $[-6.77, -0.20]$. That model finds legal moves more often and finishes proofs less often. The pattern is not consistent across blocks: at Small seed 1, canonical names lower the valid and progress rates by 8.61 and 7.79 points, and in the other two blocks these rates move by at most 1.02 points.

Table 9: Surface I effects on miniF2F, averaged over R, in percentage points. Root indicators ask whether at least one of the first eight candidates has the stated property.

Size	Seed	Valid	Progress	Root complete	Solve@64
Small	0	$+1.02$	$+0.41$	-1.43	-1.02
Small	1	-8.61	-7.79	-6.97	-7.58
Medium	0	-0.61	-0.82	-5.74	-4.92
Medium	1	$+7.79$	$+5.74$	-3.69	-3.48

B.4 OVERLENGTH STATES DURING SEARCH

Table 10 counts states that exceed the 4,096 token encoder limit during search. An overlength state consumes an expansion and is discarded. Surface cells record no overlength root or intermediate state on any benchmark. Intermediate counts cover only states selected for expansion, because successors that are never selected are never encoded.

Table 10: Overlength states in core syntax search, over all theorems. Each block has four core cells, so each denominator is four times the benchmark size. Searches affected select at least one overlength intermediate state. The last column divides rejected intermediate states by all selected intermediate states.

Benchmark	Size	Seed	Roots over limit	Searches affected	Selected states rejected
Mathlib	Small	0	202 / 2,048	9.13%	5.44%
Mathlib	Small	1	202 / 2,048	9.72%	5.14%
Mathlib	Medium	0	202 / 2,048	8.98%	3.98%
Mathlib	Medium	1	202 / 2,048	11.28%	5.27%
miniF2F	Small	0	8 / 976	2.97%	1.06%
miniF2F	Small	1	8 / 976	5.02%	2.54%
miniF2F	Medium	0	8 / 976	6.56%	4.12%
miniF2F	Medium	1	8 / 976	3.69%	1.89%
ProofNet	Small	0	66 / 1,468	7.83%	4.11%
ProofNet	Small	1	66 / 1,468	8.65%	3.80%
ProofNet	Medium	0	66 / 1,468	8.04%	2.99%
ProofNet	Medium	1	66 / 1,468	10.29%	4.43%

B.5 RELATIONS WITH SURFACE AND CORE INPUTS

Table 11 gives the R effect separately for surface and core inputs, averaging over I, and the difference between them. The difference is twice the standard $C \times R$ factorial interaction.

Table 11: R effects by syntax. Top: test token NLL in units of 10^{-3} nats per token; negative values mean lower loss with R. Bottom: $\mathrm{Solve@64}$ difference, core R effect minus surface R effect, in percentage points. Brackets give 95% paired bootstrap intervals.

Size	Seed	Surface	Core	Difference
Small	0	-0.406 [-0.583, -0.229]	-0.653 [-1.213, -0.073]	-0.247 [-0.817, $+0.371$]
Small	1	$+0.338$ [$+0.169$, $+0.513$]	-0.233 [-0.450, -0.015]	-0.571 [-0.844, -0.299]
Medium	0	$+0.170$ [$+0.011$, $+0.330$]	-0.796 [-1.339, -0.243]	-0.967 [-1.555, -0.375]
Medium	1	$+0.183$ [$+0.029$, $+0.346$]	$+0.003$ [-0.531, $+0.508$]	-0.179 [-0.741, $+0.328$]

Size	Seed	Mathlib	miniF2F	ProofNet
Small	0	-0.59 [-1.46, $+0.20$]	-1.64 [-4.30, $+1.02$]	-0.41 [-1.09, $+0.27$]
Small	1	-0.20 [-0.88, $+0.49$]	-0.20 [-1.64, $+1.02$]	0.00 [-0.54, $+0.54$]
Medium	0	$+0.20$ [-0.78, $+1.17$]	$+0.61$ [-1.84, $+3.07$]	0.00 [-0.68, $+0.68$]
Medium	1	-0.49 [-1.07, 0.00]	$+2.25$ [$+0.20$, $+4.51$]	$+0.27$ [-0.41, $+0.95$]

B.6 SEARCH WITH A LENGTH NORMALIZED FRONTIER

To test whether unnormalized scores disadvantage the longer canonical tactics, we reran miniF2F search for the eight 000 and 100 checkpoints with one change: the frontier orders states by the path log probability divided by the number of path tokens, counting each action's end token. The beam, checkpoints, benchmark population, Lean worker, budgets, and parallelism match the original evaluation. Because the beam is unchanged, every expansion proposes the same candidates as before; on the first 20 problems of one run, the rerun reproduced every original root candidate and outcome. Table 12 compares the two orderings. This analysis was added after the original results were available.

We first also ranked completed beam candidates by log probability per token. That variant favored long repetitive tactics: on one problem, the solving candidate `grind` was displaced by rewrites

Table 12: miniF2F Solve@64 under the original and the length normalized frontier. Solves are given as 000 / 100 out of 244. Effects are 100 minus 000 in percentage points, with 95% paired bootstrap intervals over theorems.

Size	Seed	Original	Normalized	Original I	Normalized I	Change
Small	0	51 / 47	50 / 45	-1.64	-2.05 [-5.74, $+1.64$]	-0.41 [-2.05, $+0.82$]
Small	1	59 / 39	58 / 39	-8.20	-7.79 [-11.89, -4.10]	$+0.41$ [0.00, $+1.23$]
Medium	0	63 / 51	63 / 49	-4.92	-5.74 [-9.02, -2.46]	-0.82 [-2.05, 0.00]
Medium	1	69 / 59	69 / 59	-4.10	-4.10 [-7.79, -0.81]	0.00 [0.00, 0.00]

repeating `one_mul`, up to 62 tokens long. After 54 to 60 problems in each of the four seed 1 runs, it had solved none, while the original search solved 2, 3, 2, and 0 of the same problems, so we stopped it and do not report it as a result.

B.7 Paired uncertainty

For tactic losses, we resample whole canonical proposition clusters. Every draw keeps all states within a sampled cluster and all conditions for each state together. We recompute the sum of losses divided by the sum of tokens for token NLL, or by the number of actions for complete action NLL. This preserves their different denominators under I. For search, we resample benchmark theorem records and retain their paired cell outcomes. All reported intervals are percentile intervals from 2,000 draws, using deterministic random streams based on seed 20260916. Input length strata, expansion ratio quartiles, and joint root accessibility are defined from the inputs before comparing the outcomes.

C Interventions on fixed checkpoints

C.1 Renaming local identifiers

The renaming sample is selected by a domain separated hash from the test states whose original and canonical action bytes are identical. Of 256 selected states, 250 meet the intervention's checks: two contain no eligible local and four do not reproduce the frozen original input tokens. The 250 retained states belong to 243 proposition clusters. Renaming changes the user names of eligible free variables in a copied Lean state while retaining their identities and dependency structure. A local is eligible when its name is an ordinary identifier: a Latin letter or one of α to δ, followed by letters, digits, or underscores. Names with primes or inaccessible markers keep their names. Aliases come from a fixed pool of conventional names: the Latin letters other than `l` and `o`; `ha`, `hb`, `hc`, `hd`, `hp`, `hq`, `hs`, `ht`, `hxy`, `hij`, `hfg`, `xs`, `ys`, `zs`, α, β, and γ; and each of these followed by a number from 1 to 20. For each eligible name, we choose the pool name not yet used in the state whose token count, both alone and after a space, is closest to the original's, breaking ties by a SHA256 hash of the transition, old name, and new name. The rule ignores the local's role, so a type variable can receive a hypothesis style name. The 250 retained states rename 1,157 locals, from 1 to 16 per state, and 974 aliases have exactly the original token counts. The remaining length change comes mostly from long names, such as `eq_add_pi_of_two_zsmul_eq_of_sign_eq_neg`, that no short alias matches. Gold action bytes stay fixed, the before state semantic hash is preserved, and execution of the gold tactic reaches the matching successor under both variants.

The same states are evaluated with the eight 000 and 100 checkpoints. Each original and renamed pair uses the same padding shape. Table 13 reports both the penalty within the original name model and the absolute comparison with its canonical counterpart. Renaming changes the original surface input length by an average of -2.768 tokens. The subset with exactly equal input token counts contains 73 states in 69 clusters; Figure 3 reports this subset separately.

For 100, the canonicalized input token sequence is identical before and after renaming on every eligible state. Its action loss is therefore identical. A generation check on 32 selected eligible states also finds identical canonical beam tokens, scores, and decoded actions. It checks eight candidates for each of eight models under both variants, giving 4,096 Lean candidate checks. The equality in Equation 5 concerns the canonical action distribution; the inverse name map then interprets that action in each corresponding live state.

Table 13: Renaming effects in nats per complete action. The penalty is renamed minus original NLL within 000, with a 95% paired cluster interval. The final two columns give 100 minus 000 NLL under each input variant.

Size	Seed	Renaming penalty	I gap, original	I gap, renamed
Small	0	0.710 [0.517, 0.928]	0.984	0.274
Small	1	0.900 [0.678, 1.132]	0.945	0.044
Medium	0	0.850 [0.627, 1.089]	1.102	0.252
Medium	1	0.821 [0.595, 1.049]	1.220	0.399

C.2 Removing and relocating relation edges

The relation sample comprises 1,024 test states in 899 proposition clusters, selected deterministically before running the interventions. We evaluate the eight 001 and 011 checkpoints. For each checkpoint and state, three passes use the real graph, an empty graph, and a relocated graph. The relocation applies a bijection to the set of distinct token endpoints and uses that same map throughout the graph. It preserves directed topology, feature labels, multiplicities, and the number of unique pairs, while changing which token positions receive the relations. Padding, text, targets, and weights are identical across the three passes. Because relations enter only encoder self attention, the passes differ only in those attention logits.

Table 14 gives the numerical contrasts and learned bias magnitudes. The mean fraction of token pairs carrying projected edges in this sample is approximately 0.205% for surface inputs and 0.0328% for core inputs. Core inputs have more edges but many more possible token pairs. We measure changes in action likelihood here; we do not rerun proof search with altered relations.

Table 14: Relation interventions and learned parameters. Loss changes are intervention minus real relations in units of 10^{-5} nats per token. The last two columns summarize absolute relation bias weights over layers, heads, and features. Figure 4 gives the intervals.

Size	Seed	C	Removed	Shuffled	Mean $\lvert b\rvert$	Max $\lvert b\rvert$
Small	0	0	-1.294	-0.797	0.0370	0.3032
Small	0	1	-0.048	-0.008	0.0201	0.1671
Small	1	0	$+1.299$	$+1.374$	0.0314	0.1950
Small	1	1	-0.264	-0.269	0.0174	0.1465
Medium	0	0	-0.365	-0.053	0.0196	0.1793
Medium	0	1	-0.326	-0.351	0.0100	0.0929
Medium	1	0	-0.560	-1.041	0.0223	0.2005
Medium	1	1	-0.022	-0.025	0.0110	0.1133

D PAIR: exposing local proof structure

PAIR stands for Proof State Alignment and Instantiation Residuals. It extends the canonical surface representation with compact input records describing typed structural alignments, local rule instantiations, and their remaining obligations. Its starting point is cell 101. The original tokens form a protected prefix, and the existing relation projection and action target remain unchanged. The compiler reads the legal before state and its local context. It never reads the action or successor.

The compiler produces two kinds of record. An alignment record compares a local hypothesis type with a goal target, or the two sides of an `Eq` or `Iff` target. It gives their shared structure with typed holes at the differing subterms and names any local equality or equivalence that bridges a hole. An instantiation record opens up to 16 leading binders of a local rule, matches its conclusion against a target, and gives the substitution and the remaining obligations. On a test fixture with `l5 : ∀ z, l2 z → l3 (l1 z), l6 : l2 l4`, and target `l3 (l1 l4)`, the compiler emits the following record, which is one line in the input and is wrapped here:

```
inst g0 l5>target sigma[p0=fvar(l4)]
   delta[p1:default:proof:app(fvar(l2),fvar(l4)) support=l6]
```

The record states that instantiating the first binder with `l4` leaves one proof obligation, supplied by `l6`. Record fragments reuse the core grammar of Appendix A. Matching uses reducible transparency and treats existing metavariables as rigid. Each state receives at most 16 records within 50,000 heartbeats, and records are admitted whole in a fixed priority order.

For a base input of L tokens, including its boundary tokens, the suffix budget is

$$B(L) = \min\big(512,\ 128 + \lfloor L/2 \rfloor,\ 4096 - L\big). \tag{6}$$

The adapter admits complete self contained records within this budget. If no record fits, it returns the exact 101 input. This preserves the entire shared population. The materialized corpus contains annotations for 7,253 of 153,129 states: 6,533 training, 381 validation, and 339 test states. It adds 1,269,802 encoder tokens. The remaining 145,876 states keep the exact 101 input. A missing record means that the bounded compiler found no alignment or instantiation within its limits, not that none exists. Compiler coverage is recorded as partial on 35,359 states, where a probe reached a resource limit or an unsupported construct; these states retain their ordinary base input and any admitted records.

The four completed PAIR runs use the corresponding 101 initialization, action targets, example schedule, and 12,927 updates. Microbatch membership and decoder padding remain matched. Encoder padding differs in two microbatches for Medium seed 0 and in one microbatch for each seed 1 run; the Small seed 0 plan is identical. Live proof search uses the same annotation compiler and token allowance.

Table 15: The four PAIR runs and their matched 101 references. NLL is measured on 7,616 states; proof counts use 64 expansions and the same benchmark populations as Table 1.

Size	Seed	Input	NLL	Mathlib /512	miniF2F /244	ProofNet /367
Small	0	101	2.235360	74	51	21
Small	0	PAIR	2.236129	74	42	19
Small	1	101	2.230382	69	41	20
Small	1	PAIR	2.231481	71	47	20
Medium	0	101	2.090563	72	50	20
Medium	0	PAIR	2.094994	80	59	20
Medium	1	101	2.099435	77	61	22
Medium	1	PAIR	2.100136	77	60	20

With PAIR, we match or exceed the 101 Mathlib solve count in each block. We find the largest gain for Medium seed 0: eight additional Mathlib proofs and nine additional miniF2F proofs. The other blocks show smaller changes and varying external benchmark outcomes (Table 15). These four runs remain exploratory.

E Examples from saved proof paths

We selected these examples from the saved deterministic search traces. We first identify theorem outcomes that differ between 000 and a single enabled factor at both seeds, then choose the smallest domain separated SHA256 hash within each benchmark, size, factor, and direction.

Canonical local hypotheses. Consider `Filter.HasBasis.eq_of_same_basis`. Both Small 100 models prove this theorem; the corresponding 000 models leave it open within the budget. The saved canonical paths are:

```
Seed 0:  ext
   simp [l7.mem_iff]
   simp [l6.mem_iff]

Seed 1:  ext
   rw [l6.mem_iff, l7.mem_iff]
```

The local identities remain usable in executable tactics. Simplification and rewriting provide two paths through the same local facts.

A recurring core syntax solve. Both Small 010 models prove `Set.Ico_insert_right` using the same four search actions:

```
ext
simp
simp [lt_iff_le_not_ge]
grind
```

The corresponding 000 runs leave the theorem open, despite solving more theorems overall.

A recurring surface solve. All four 000 models prove `Real.sin_antiperiodic`. Both Medium models use:

```
intro x
simp [sin_add]
```

The Small paths also finish with `simp [sin_add]`, following an initial simplification and, at seed 1, an introduction. All four 100 models leave this theorem open within the search budget. The useful signal here includes the choice of a named global trigonometric lemma, which is present in every representation.

An external induction proof. All four 100 models prove ProofNet Verified record 212, `Dummit_Foote_exercise_1_1_15`; the 000 models leave it open. Both Medium models and Small seed 1 use the single action

```
induction l2 <;> simp [*]
```

Small seed 0 instead uses `simp`, then `induction l2 <;> simp`, then `simpa`. All four external paths have saved successful compilation and kernel verification records.