Before Lean Checks: Candidate Exposure in Proof-Action Ranking

Shivangi Kamat [1] **Yangshuai Wang** [2]

Abstract

Formal proof agents do not fail only when they cannot finish a theorem; they can fail earlier, when ranking hides the tactics Lean would have accepted. This paper studies that pre-check handoff: which proof-state representations and family priors keep useful candidate tactics inside the short list sent to Lean? We separate inputs visible before the next action from future-tactic metadata, then compare unguided retrieval, hard family routing and a soft family prior on a curated mathlib4 subset. Trace matches suggest that soft family guidance is competitive with unguided retrieval and that hard routing loses alternatives. A 500-state Lean check gives the more consequential picture: unguided retrieval has the strongest top-five Lean acceptance, while soft guidance is close but not better. The result reframes family prediction as a ranking-control problem: useful proof agents should surface several legal next moves for Lean, not merely imitate one recorded trace step.

1. Introduction

Formal proof assistants make feedback for proof agents concrete: a proposed action can be checked by Lean (de Moura & Ullrich, 2021), Coq or Isabelle rather than judged only by surface plausibility. This verifier-backed setting underlies CoqGym, LeanDojo/ReProver and recent formal-mathematics language-model systems (Yang & Deng, 2019; Yang et al., 2023; Polu & Sutskever, 2020; Polu et al., 2023). It matters for mathematical agents because Lean can provide feedback only on actions that the learned system actually sends for checking.

Many proof-agent systems focus on model scale, search width, premise retrieval or tactic generation (Gauthier et al., 2021; Loos et al., 2017; Kaliszyk et al., 2018; Lample et al., 2022). The proof state itself often enters as whatever text or metadata the environment happens to expose. In a verifier-backed loop, that choice is consequential. If a representation hides useful hypotheses, available premises or goal shape, the system may never place a verifiable action near the top. If a representation overfits to incidental names and formatting, it may learn surface regularities that fail on new theorems.

We call this handoff *candidate exposure*: whether a learned ranking places Lean-acceptable actions inside the short list that will be checked. Classification accuracy asks whether a representation predicts a coarse action type; similarity retrieval asks whether a candidate resembles a past proof step; exposure asks whether those signals deliver useful actions to Lean.

This framing matters beyond the small family-prior method used here. A proof agent that never exposes legal alternatives receives no accepted branches and no Lean feedback for repairing local identifiers or tactic arguments, so exposure decides whether verifier feedback can reach the next round of modeling.

A Lean proof state contains the target proposition, local hypotheses, types, dependencies, notation, library references and syntactic structure. Treating its pretty-printed text as a neutral input risks confusing proof-relevant signal with formatting, local names or incidental notation. Replacing the state with handcrafted summaries may remove useful context. The representation choice is therefore part of the proof-search design.

There is also a sharper availability rule. Before the next tactic is chosen, a prover may see the current goal, local context, theorem name, file path, imports and premises retrieved from the library. The prover has not yet seen the premises that the next tactic will actually use, and it has not seen the syntax tree of that future tactic. Those fields are easy to extract from a replayed trace, but using them as input changes the task into hindsight prediction. We therefore keep them separate throughout the paper.

We make this question concrete through tactic-family prediction. Instead of immediately generating a complete tactic string, the learner predicts a coarse family such as `intro`,

[1]School of Informatics, University of Edinburgh, Edinburgh, United Kingdom [2]Department of Mathematics, National University of Singapore, Singapore. Correspondence to: Yangshuai Wang <yswang@nus.edu.sg>.

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).

`rw`, `simp`, `exact`, `apply`, `cases` or `constructor`. This is the level at which many proof agents make their first routing decision: introduce variables, rewrite, simplify, split a goal, apply a theorem or close by assumption. We then use Exposure-Preserving Family Prior (EPFP) as a small test of how family information changes the list sent to Lean. EPFP softly reorders retrieved candidate tactics while keeping off-family alternatives available for checking.

The study has four parts. First, it treats candidate exposure as the handoff between ranking and Lean checking. Second, it separates state-visible representations from future-tactic premises and tactic AST metadata. Third, it uses EPFP as a compact family-prior example: soft guidance changes rankings without discarding off-family candidates, whereas hard family gates can hide plausible tactics. Finally, it checks the ranked candidates in Lean on held-out intermediate states and compares that result with trace and family matches.

The experiments follow the handoff itself: choose a state representation, use family predictions to reorder retrieved tactics, and then check the resulting lists in Lean. Figure 1 summarizes this path and the fields kept out of the main inputs.

2. Related Work

Learning-guided proof search. Learning-guided theorem proving has long connected proof states to action ranking. TacticToe learns tactic-level guidance for HOL4 and uses it inside Monte Carlo tree search (Gauthier et al., 2021). Deep network-guided proof search showed that learned guidance can improve first-order automated theorem proving (Loos et al., 2017). Reinforcement-learning systems such as rlCoP learn from previous proof attempts to guide large-scale proof search (Kaliszyk et al., 2018). GamePad studies proof-step prediction and proof-state evaluation in Coq (Huang et al., 2019). These systems demonstrate the value of learned search guidance; the present paper focuses on the representation-to-exposure step before the search policy consumes it.

Lean environments and theorem-proving datasets. Proof-assistant datasets and environments provide the substrate for this kind of study. CoqGym provides a Coq proof environment and ASTactic model for generating Coq tactics as abstract syntax trees (Yang & Deng, 2019). HOList introduced a higher-order theorem-proving environment for machine learning over HOL Light (Bansal et al., 2019), while miniF2F emphasizes cross-system formal mathematical problem solving (Zheng et al., 2021). Lean's mathematical library makes large-scale Lean studies possible (The mathlib Community, 2020). Lean-based systems show that proof traces can improve language-model theorem proving (Han et al., 2021), and LeanDojo/ReProver provides tools for retrieval-augmented theorem proving, premise selection and interaction with Lean proof states (Yang et al., 2023).

Language models and premise selection. Language-model theorem provers such as GPT-f frame formal proof search as action generation against a verifier (Polu & Sutskever, 2020); expert-iteration and curriculum approaches use proof search itself to generate new training signal (Polu et al., 2023); and HyperTree Proof Search connects neural proposal models with online search across formal environments (Lample et al., 2022). Draft, Sketch, and Prove uses informal proof sketches to guide formal proving (Jiang et al., 2023). Recent Lean 4 work also studies statement autoformalization and theorem retrieval, including ATLAS (Liu et al., 2025), Mathlib4 semantic search (Gao et al., 2024) and LeanSearch v2 (Gao et al., 2026). Premise selection is a related hard part: DeepMath studies neural premise selection for large formal corpora (Irving et al., 2016), while LeanDojo/ReProver combines premise retrieval with Lean interaction (Yang et al., 2023). Our representation protocol follows the same availability principle: retrieved premises may be used only when they are selected from information available before the next tactic, whereas annotated premises from the target tactic are future information.

Trace supervision and verifier feedback. Learning-to-rank methods such as RankNet optimize ranking functions from supervised signals (Burges et al., 2005). In proof-action ranking, the supervised signal is often an exact match to the traced tactic, but the verifier accepts a wider set of valid continuations. The same distinction applies to exploration in verifier-backed agents: a ranking method can be evaluated by both offline trace matches and the candidates Lean accepts.

Our study narrows this line of work to the last ranking step before a tactic reaches Lean, where trace matching and direct Lean checking can be compared on the same candidate lists.

3. Problem Setup

Verifier-backed proof-action ranking. A proof assistant gives a sharp feedback signal. A proof-action ranking method proposes a short list of candidates, and Lean determines which, if any, are valid continuations. A learned system rarely searches over all possible tactic strings. It first maps a proof state s to a representation $\phi(s)$, scores a candidate set $C(s)$, and only then sends a few candidates to the verifier. Representation is therefore part of the search policy: it helps determine which proof information can affect the actions Lean sees.

This motivates candidate exposure. For a ranked candidate

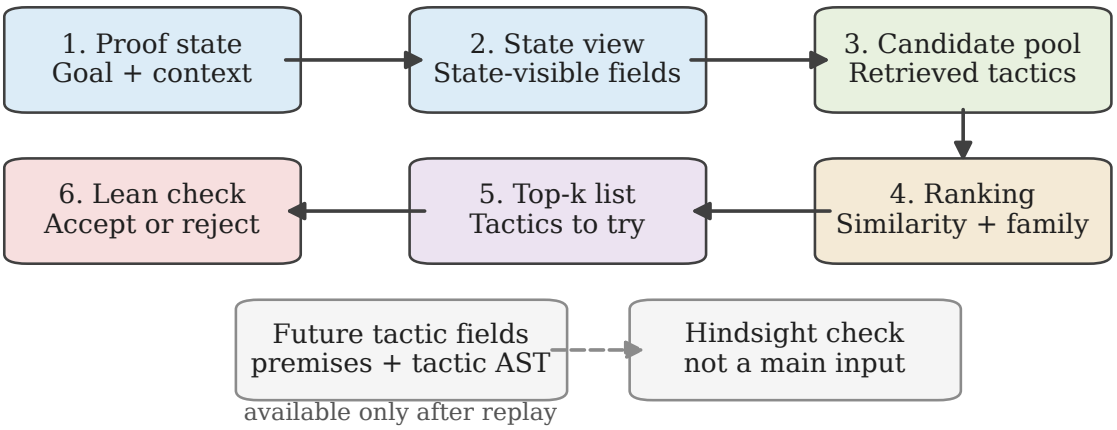

Figure 1. Candidate path before Lean checking. The main experiments use only state-visible inputs; future-tactic premise and AST fields are kept as hindsight-only comparisons.

list, exposure asks whether a Lean-accepted tactic appears among the top k candidates. Large LeanDojo traces record the tactic that appeared in the proof while omitting other alternatives Lean might have accepted. We keep trace matches, family matches and Lean acceptance as separate measurements. This connects representation learning to verifier feedback: Lean can accept an action only after it appears in the checked list.

A trace match is a narrow event. A proof may record `rw [h]`, while `simpa [h]` or an `exact` proof term may also be legal in the same state. The reverse can happen after transfer: a candidate can look textually close to the trace but fail because a local identifier, implicit argument or type-class instance is reconstructed differently. For this reason, the execution section checks the same ranked lists in Lean rather than treating a trace miss as a useless candidate.

In the retrieval study, an action a has its own stored training state s_a and family $f(a)$. Unguided retrieval uses a similarity score $\mathrm{sim}(\phi(s), \phi(s_a))$. EPFP adds a family prior:

$$r_\lambda(s, a) = \mathrm{sim}(\phi(s), \phi(s_a)) + \lambda\, p_\theta(f(a) \mid \phi(s)),$$

where p_θ is the tactic-family model and λ controls how much family information can reorder candidates. EPFP keeps every candidate rankable by similarity; family scores only perturb the order. Hard family guidance is a lexicographic gate: candidates from the top predicted family are ranked before candidates from lower predicted families, regardless of small similarity differences. True-family guidance is included as a ceiling.

Why this layer matters for AI4Math. Self-improving mathematical agents need a reliable feedback loop: propose actions, verify them, learn from successes and failures, and improve future rankings. Candidate exposure is the point where this loop can start or stall. Rankings that bring plausible actions forward give the agent accepted actions, rejected actions and Lean error messages to learn from. The representation-to-ranking layer tells system builders whether the failure happens before Lean is allowed to judge the action.

4. Task and Dataset

Proof-step examples. The unit of analysis is a checked Lean proof step immediately before a tactic is applied. For each LeanDojo theorem file, the dataset records the source location, theorem name, step index, main goal, local context and optional metadata:

$$x_i = (\mathrm{file}, \mathrm{theorem}, \mathrm{step}, \mathrm{goal}, \mathrm{context}, \ldots).$$

The target is the family of the next tactic, $y_i = \mathrm{family}(\mathrm{next_tactic}_i)$. The full tactic string is retained for next-action retrieval and execution checks.

Why tactic families. Full tactic prediction is a structured-generation problem involving theorem names, rewrite lists, local identifiers and nested proof terms. Tactic families expose the first routing decision in that problem: does the represented state contain enough visible signal to choose the kind of next proof move? The abstraction is still nontrivial because similar state text may support `rfl`, `rw`, `simp`, `ring` or `exact`, depending on local context and available facts.

Controlled mathlib4 subset. The main dataset is mixed enough to stress first-stage routing while staying small enough to inspect. It combines the earlier S3 arithmetic,

mixed-library and proof-dense subset with four S4 dense-core modules chosen by source-level screening and small-batch trace checks. The resulting collection has 3,723 proof steps across 46 traced files and 81 tactic-family labels. The label distribution is challenging: `rw`, `simp`, `exact` and `simpa` dominate, leaving a long tail of rare families. An S4-E extension adds two dense-core modules and is kept in the appendix.

Theorem-level splits. All reported splits are by theorem name, not by individual proof step. This prevents adjacent steps from the same proof from appearing in both training and test sets, which would leak local identifiers, intermediate facts and proof style. The protocol is stricter than random step-level splitting and better matches the intended use case: ranking actions for theorems not seen during training.

State-visible premise information. Premises are useful only if they are obtained in the same order a prover would obtain them. Retrieved-premise features are built by searching the training corpus with the current proof state and then appending the returned premise names. They are allowed because the retriever uses only information available before the next action. Future-premise features come from the recorded tactic itself; they answer a different question, namely how much family signal would be present if premise selection and tactic-argument discovery were already solved. This distinction is syntactically small but interpretively large.

5. Method

Overview. The method has two layers. First, we materialize several representations of the same checked proof states and train lightweight tactic-family predictors under by-theorem splits. Second, we attach the learned family distribution to retrieval through EPFP, which is designed to preserve candidate exposure rather than gate candidates away. This ordering separates two questions: classification measures whether a representation carries family information, while EPFP asks whether that information helps Lean see useful actions. EPFP is deliberately small: learned family information affects priority while keeping plausible alternatives visible to the verifier. Comparing EPFP with unguided retrieval, hard family gates and true-family guidance shows when a family signal helps the ranking and when a discrete routing decision blocks it.

Representations. We compare proof-state views that differ in what is available before the next tactic is known. Raw and normalized text preserve the pretty-printed goal and local context as exposed by Lean, with normalized text removing selected names, numbers and whitespace variation. Structured summaries keep shallow goal features such as goal shape, hypothesis count, token counts, connective counts, arithmetic-operator counts and coarse head symbols. State-metadata views add file, module and theorem identifiers. Retrieved-premise views add premise names retrieved from the training corpus using the current proof state; they never read the held-out step's gold tactic premises. Future-premise views use the next tactic's annotated premises and tactic AST metadata and are reported only as a ceiling.

The state-metadata view is intentionally modest. File and theorem names can encode domain, notation and local proof idiom, but they also tempt memorization of nearby library style. We use theorem-level splits as the main guard and keep file-level, duplicate and grouped-family supporting checks in the artifact.

Leakage check. LeanDojo can annotate a tactic with the premises it actually used and with the syntax tree of that tactic. Those fields become available only after the next tactic is known. All deployable representations therefore exclude `premises`, `annotated_tactic`, `ast_summary`, `state_after` and `next_tactic`. The former premise-aware representation is renamed future-premise in the paper logic and kept out of the main conclusions.

Models. We use small classifiers: majority class, a keyword heuristic, bag-of-words Naive Bayes, TF-IDF logistic regression and TF-IDF linear SVM. This keeps the representation comparison readable: each proof-state view is evaluated with the same model families, splits and targets.

Evaluation protocol. All splits are by theorem name, so proof steps from the same theorem never appear in both training and test sets. For tactic-family prediction, accuracy reflects common families, macro-F1 exposes rare-family weakness, and top-k accuracy measures whether the correct family remains visible beyond the first guess.

We then ask how that information changes the actions that Lean would see. For each held-out proof state, training-set tactics form the candidate set and unguided retrieval ranks them by TF-IDF similarity. EPFP adds the learned family prior from the setup above; hard family guidance puts all candidates from the single predicted family ahead of all others; top-m and reciprocal-rank-fusion variants provide less brittle family-aware baselines; true-family guidance gives a ceiling. EPFP weights are selected on a validation split. The ranking tables track trace matches, family matches and the mean rank of the first true-family candidate.

We keep three measurements separate. Trace and family match ask whether a retrieved candidate equals the recorded next tactic or shares its family. Lean applicability asks whether Lean accepts a transferred candidate as a legal one-step tactic in a reconstructed intermediate state. Full

LeanDojo proof search would execute tactics in the original interactive environment across multiple search steps; here we focus on the one-step check.

The behavior is direct. When the top predicted family $\hat{y}$ is wrong, a hard gate can fill the top-k list with candidates from that wrong family and hide a nearby candidate that Lean might accept. EPFP avoids this all-or-nothing behavior because every candidate remains rankable by similarity and family information only changes priority. For two candidates a, b, EPFP ranks a above b when

$$\Delta_{\text{sim}} + \lambda \Delta_p > 0,$$

where Δ_{sim} is their similarity gap and Δ_p is their family-probability gap. Since $\Delta_p \in [-1, 1]$, candidates separated by more than λ in similarity keep their order. The family prior therefore acts as a local preference when λ is small, rather than as a rule that removes alternatives.

To connect this layer to verification, we check ranked tactics on held-out mathlib states and record each $(\text{query}, \text{strategy}, \text{rank}, \text{tactic})$ item sent to Lean. This lets us compare trace matching, family matching and Lean acceptance for the same candidate lists.

6. Results

Dataset shape. The dataset is a compact routing set. Its 3,723 proof steps cover many theorem names and 81 tactic families, with most probability mass in a few common families. Head-family accuracy is therefore easy to improve by exploiting common tactics, while macro-F1 reflects how little data exists for many rare proof moves.

RQ1: state-visible representations carry family signal. The first result is that representation changes the proof signal visible to even small models. The leakage-checked study separates deployable state-only and retrieved-premise inputs from the future-premise ceiling. Inputs that use the next tactic's annotated premises or tactic AST are best read as hindsight information.

Here we report by-theorem results; the artifact contains file-level, per-family and rare/mid/head supporting checks. State-visible models recover head-family signal, but macro-F1 remains low, confirming that rare proof moves are still difficult with shallow state text. Future-premise numbers estimate the headroom left for premise selection and tactic-argument generation.

The deployable conclusion therefore rests on state-visible features and on premises retrieved without reading the held-out tactic.

This pattern separates head-family signal from long-tail weakness. The head families are learnable from visible state text; the tail remains weak; and premise-bearing features help only when the premises are obtained by a legal retrieval step. The path for improvement is therefore concrete: better premise retrieval and argument construction are likely to matter more than adding another shallow classifier on the same pretty-printed state.

Table 1. Controlled LeanDojo subset characteristics. The dataset is extracted from a curated mathlib4 subset rather than a full-mathlib trace.

Statistic	Value
Proof steps	3723
Theorems	1702
Tactic families	81
Mean steps/theorem	2.187
Mean goal tokens	11.084
Mean context size	5.784

RQ2: soft family priors are safer than hard gates. The main ranking result is that hard symbolic routing is brittle. In mixed proof-step retrieval, an early family mistake can hide otherwise similar candidates. Across by-theorem splits, hard gating pushes the first true-family candidate from roughly rank 41 to above rank 500, while validation-selected EPFP stays near unguided retrieval. Its exact@5 is essentially tied with unguided retrieval. The true-family row clarifies the ceiling: even knowing the true family leaves within-family premise and argument selection as the next problem.

Validation selection also prevents over-reading the family prior. The selected EPFP weights vary across splits, and the paired tests show only a small exact@1 gain. The mechanism is easy to see: when the top family is wrong, hard gating displaces the true family by hundreds of ranks, while EPFP usually moves it by only a few positions. Uncertain family predictions work better as soft preferences than as filters that delete plausible alternatives before Lean can check them.

Family prediction is most useful here as a soft ranking signal; trace-match gains are only the first part of the story.

RQ3: Lean checking changes the picture. With Lean 4.28, we check the top five tactics from each strategy on 500 held-out S4 intermediate states. Almost all candidates run normally; the remaining failures are state-reconstruction issues, mainly unresolved local identifiers or type-class elaboration after tactic transfer.

Trace matching and Lean checking answer different questions. Trace matching asks whether a retrieved tactic is the one recorded in the proof. Lean checking asks whether the tactic is a legal one-step continuation in the reconstructed state. Lean often accepts alternatives to the trace tactic,

Table 2. Tactic-family prediction results under by-theorem splitting. Accuracy, macro-F1 and top-3 accuracy compare the same model families across proof-state representations. Bold marks the strongest state-visible entry in each metric; future premise is a hindsight comparison.

Representation	Model	Accuracy	Macro-F1	Top-3
State-only	Majority	0.262	0.007	0.495
	Keyword	0.168	0.015	0.449
	Naive Bayes	0.335	0.065	0.588
	TF-IDF LR	0.109	0.066	0.245
	TF-IDF SVM	0.214	0.096	0.456
State+metadata	Majority	0.262	0.007	0.495
	Keyword	0.168	0.015	0.449
	Naive Bayes	**0.352**	0.063	0.592
	TF-IDF LR	0.117	0.073	0.252
	TF-IDF SVM	0.217	0.099	0.456
Retrieved-premise	Majority	0.262	0.007	0.495
	Keyword	0.168	0.015	0.449
	Naive Bayes	0.337	0.073	**0.594**
	TF-IDF LR	0.124	0.072	0.278
	TF-IDF SVM	0.237	**0.103**	0.469
Future premise	Majority	0.262	0.007	0.495
	Keyword	0.168	0.015	0.449
	Naive Bayes	0.360	0.051	0.586
	TF-IDF LR	0.130	0.076	0.288
	TF-IDF SVM	0.296	0.124	0.546

Table 3. Split-seed stability on the controlled subset. Values are mean $\pm$ population standard deviation over six by-theorem random splits. Bold marks the strongest state-visible mean in each metric.

Representation	Model	Accuracy	Macro-F1	Top-3
Future premise	Naive Bayes	0.352 ± 0.004	0.051 ± 0.005	0.589 ± 0.012
	TF-IDF SVM	0.301 ± 0.009	0.129 ± 0.008	0.547 ± 0.020
Retrieved-premise	Naive Bayes	0.329 ± 0.007	0.073 ± 0.006	0.585 ± 0.008
	TF-IDF SVM	0.240 ± 0.016	0.097 ± 0.010	0.469 ± 0.016
State+metadata	Naive Bayes	$\mathbf{0.338} \pm 0.010$	0.059 ± 0.004	$\mathbf{0.596} \pm 0.014$
	TF-IDF SVM	0.229 ± 0.012	$\mathbf{0.101} \pm 0.008$	0.449 ± 0.020
State-only	Naive Bayes	0.329 ± 0.009	0.066 ± 0.007	0.586 ± 0.013
	TF-IDF SVM	0.219 ± 0.010	$\mathbf{0.101} \pm 0.010$	0.449 ± 0.016

including different simplification, rewrite, closure or construction moves.

Table 5 and Figures 2 and 3 change the ranking-only story. Soft family guidance is tied with unguided retrieval on exact@5, but unguided retrieval is strongest after Lean checking: Accept@5 is 0.672 for unguided and 0.652 for soft family guidance. The paired comparison gives a -2.0 percentage-point difference for soft guidance, with an interval crossing zero (Appendix Table 7). The execution result is therefore a tie with a lower point estimate for EPFP, rather than a top-five Lean-acceptance improvement.

Accepted alternatives explain why trace matching understates what Lean can use. For unguided retrieval, 62.8% of queries have a Lean-accepted top-five tactic different from the recorded tactic, and 53.4% have Lean acceptance without any trace hit. These are one-step continuations rather than complete proofs, but they show why a trace miss can still be useful to Lean.

Retrieval matches help; Lean acceptance tells us which candidates actually run.

The accepted non-trace cases are mostly ordinary proof engineering rather than surprising proof discoveries. Many are variants of simplification, rewriting, constructor use or closure by a nearby hypothesis. This is exactly the region where a ranking method can help an interactive prover: the trace gives one path through the proof, while Lean may accept several small local moves that keep the proof moving. A ranking table that sees only the traced string throws those alternatives away.

What the Lean check reveals. The 500-state run changes the interpretation in three ways. First, exact trace matching

Table 4. Retrieval ranking before Lean checking. The table reports family-level and trace-tactic success at fixed ranks. Bold marks leading non-ceiling entries; near-equal exact@5 values are treated as ties.

Strategy	Family@1	Family@5	Exact@1	Exact@5
Unguided	**0.354**	**0.600**	0.120	**0.169**
Soft family (val-selected)	0.337	0.589	**0.123**	**0.170**
Hard family	0.113	0.130	0.069	0.082
Family RRF	0.261	0.546	0.111	**0.169**
Top-m family	0.108	0.231	0.065	0.111
True family	0.988	0.988	0.199	0.245

Table 5. Lean checking results on 500 held-out S4 states. Each strategy checks 2500 candidate tactics. Accept@k uses all queries as the denominator; coverage is the fraction of candidate tactics that reached ordinary Lean acceptance or rejection after state reconstruction. Bold marks the best value in each column.

Strategy	Accept@1	Accept@3	Accept@5	Coverage	Recon. fail
Unguided	0.424	0.588	**0.672**	**0.988**	**0.012**
Soft family	**0.428**	**0.594**	0.652	0.987	0.013
Hard family	0.424	0.516	0.580	0.980	0.020
Family RRF	0.412	0.562	0.622	0.982	0.018
Top-m family	0.372	0.558	0.590	0.964	0.036

is a conservative reading of usefulness: it rewards recovering the logged proof step, but Lean often accepts a neighboring tactic that makes progress from the same state. Second, family-aware ranking mainly changes which alternatives survive to the top five. Soft guidance preserves many of the same alternatives as unguided retrieval, while hard routing replaces them with candidates from one family and loses useful off-family moves. Third, most accepted non-trace tactics are not exotic; they are the kind of local rewrite, simplification, constructor or closure move that an interactive user might try when the traced proof is unavailable. The practical message is that a prover needs several legal next moves, not only the exact move written in mathlib.

The failures are equally informative. Unknown identifiers point to local-context transfer, not to a failure of the mathematical idea behind the tactic. Elaboration failures often arise when the reconstructed state lacks the precise implicit arguments or instances present in the original proof script. Ordinary Lean rejections, by contrast, say the candidate is simply the wrong move for that state. Separating these cases makes the execution run more useful than a single accept/reject number.

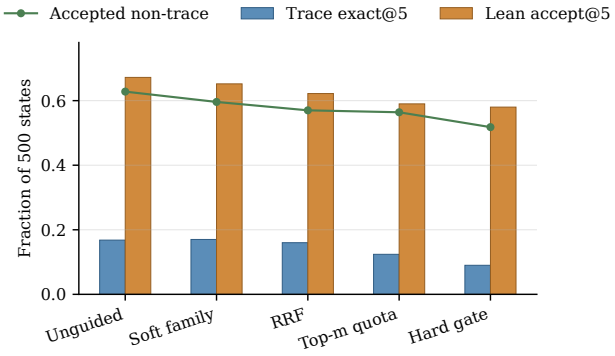

Figure 2. Top-five trace exact match and Lean acceptance on the same 500 held-out states. Lean accepts many non-trace tactics, and unguided retrieval gives the highest top-five acceptance in this run.

Errors concentrate around overloaded families. The largest confusions identify the next problem after family exposure. Most errors center on `rw`, often confused with simplification-adjacent or construction-oriented families such as `simpa`, `constructor`, `cases` and arithmetic automation. These plausible alternatives point to the next modeling layer: within-family premise selection and tactic-argument generation.

Qualitative inspection gives the same pattern. Some failures are simplification/rewrite ambiguities, others are construction/refinement ambiguities, and a third class depends on exact local premises or arguments. These errors explain why even the true-family row in Table 4 is far from perfect. The family labels are therefore best read as a first routing layer, not as the final proof action. The `rw`/`simp`/`simpa`/`rwa`/`simp_rw` group is especially important: these tactics often differ in how they combine rewriting, simplification and goal closure, but a theorem prover ultimately needs the right lemma list and local identifiers. Treating each surface family as fully separate can exaggerate some mistakes while hiding the harder argument-selection problem.

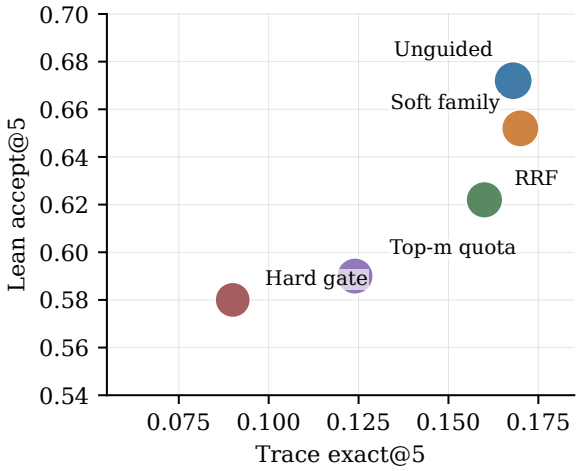

Figure 3. Trace exact@5 versus Lean accept@5 by ranking strategy. Larger markers indicate more accepted non-trace top-five tactics; all strategies have many Lean-accepted tactics that are not exact trace matches.

7. Discussion and Implications

Exposure is the handoff to Lean. The main design lesson is that proof-state representation is part of the search problem. In this dataset, state-visible representations carry usable lightweight family signal, while future-premise results show the headroom that remains when the next tactic's actual premises are known. The retrieval study favors using family predictions as ranking hints rather than unconditional filters when uncertainty is high. This matters for verifier-backed mathematical agents because correctness is established when Lean checks a candidate action, and Lean can check only actions that the ranking method exposes.

Representation errors are often silent. If a ranking hides a useful local hypothesis or overcommits to one tactic family, Lean only accepts or rejects the submitted tactics. The ranking layer should therefore be judged by the legal moves it brings within reach, not only by imitation of the traced proof.

Trace matches are a starting point. Self-evolving proof agents need inexpensive measurements, and trace or family matches are useful because they are cheap and reproducible. Lean checking adds the missing view: it shows which retrieved tactics are accepted even when they differ from the trace. When trace matches and Lean acceptance diverge, the hard part has often moved from family exposure to argument choice, premise selection, local identifier recovery or proof-script transfer.

This also changes how to read a negative ranking result. If a method improves exact trace hits but lowers Lean-accepted top-five actions, it has become more faithful to the recorded proof without becoming more useful to the checker. If it preserves Lean acceptance while changing the trace-hit rate only slightly, it may still be a good search component. For proof agents, the ranking layer is successful when it gives Lean several plausible moves, not when it imitates the single logged move at all costs.

Hard symbolic gates can damage learning-to-rank. The retrieval result turns a common engineering choice into a visible design decision. A stronger family predictor can become a worse search rule when it is used as a hard symbolic gate: its mistakes can prevent the verifier from seeing candidates that similarity retrieval would have kept near the top. EPFP is a gentler option for mixed proof-state ranking because it lets family information influence rank while leaving lower-probability families in the list.

What the 500-state Lean run adds. The execution run is small enough to inspect and large enough to change the reading of the ranking tables. It shows that the best-looking family prior under trace matching need not be the best list for Lean checking. It also shows that transferred tactics fail for understandable reasons: unresolved local names, elaboration after reconstruction, or a tactic that is simply inapplicable to the reconstructed state. These failure modes are exactly where a future prover needs tighter coupling between retrieval, local context repair and Lean feedback.

8. Limitations

The study uses a curated mathlib4 subset, coarse tactic-family labels and lightweight models. The curated subset supports manual inspection and rapid iteration; full-mathlib evaluation remains future work. Tactic-family prediction isolates first-stage action routing, while the next modeling layer is within-family premise selection, rewrite-list construction and local-identifier recovery.

The execution study measures one-step applicability in reconstructed intermediate states, not theorem-level solving. Full LeanDojo replay remains future work, as does tighter local-identifier and tactic-argument repair inside the proof-search loop.

9. Conclusion

We studied candidate exposure as the handoff between Lean proof-state representation and verifier-backed action ranking, instantiated with EPFP. On a 3,723-step mathlib4 subset, state-visible family signals and future-premise ceilings separate deployable representation quality from next-tactic leakage. The practical lesson is that representation decides which tactics Lean sees, so trace scores should be read together with the legal alternatives left in the submitted list at the verifier handoff.

References

Bansal, K., Loos, S. M., Rabe, M. N., Szegedy, C., and Wilcox, S. J. HOList: An environment for machine learning of higher order logic theorem proving. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 454–463. PMLR, 2019.

Burges, C. J. C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 89–96, 2005.

de Moura, L. and Ullrich, S. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pp. 625–635. Springer, 2021.

Gao, G., Ju, H., Jiang, J., Qin, Z., and Dong, B. A semantic search engine for Mathlib4. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8001–8013. Association for Computational Linguistics, 2024.

Gao, G., Sun, Z., Jiang, J., Wang, Y., Xu, J., Wu, P., Dai, B., and Dong, B. LeanSearch v2: Global premise retrieval for Lean 4 theorem proving. *arXiv preprint arXiv:2605.13137*, 2026.

Gauthier, T., Kaliszyk, C., Urban, J., Kumar, R., and Norrish, M. TacticToe: Learning to prove with tactics. *Journal of Automated Reasoning*, 65(2):257–286, 2021. doi: 10.1007/s10817-020-09580-x.

Han, J. M., Rute, J., Wu, Y., Ayers, E. W., and Polu, S. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021.

Huang, D., Dhariwal, P., Song, D., and Sutskever, I. GamePad: A learning environment for theorem proving. In *International Conference on Learning Representations*, 2019.

Irving, G., Szegedy, C., Alemi, A. A., Een, N., Chollet, F., and Urban, J. DeepMath - deep sequence models for premise selection. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

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

Kaliszyk, C., Urban, J., Michalewski, H., and Olšák, M. Reinforcement learning of theorem proving. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Lample, G., Lacroix, T., Lachaux, M.-A., Rodriguez, A., Hayat, A., Lavril, T., Ebner, G., and Martinet, X. HyperTree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Liu, X., Bao, K., Zhang, J., Liu, Y., Chen, Y., Liu, Y., Jiao, Y., and Luo, T. ATLAS: Autoformalizing theorems through lifting, augmentation, and synthesis of data. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Loos, S., Irving, G., Szegedy, C., and Kaliszyk, C. Deep network guided proof search. In *21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46 of *EPiC Series in Computing*, pp. 85–105. EasyChair, 2017.

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

Polu, S., Han, J. M., Zheng, K., Baksys, M., Babuschkin, I., and Sutskever, I. Formal mathematics statement curriculum learning. In *International Conference on Learning Representations*, 2023.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 367–381. ACM, 2020. doi: 10.1145/3372885.3373824.

Yang, K. and Deng, J. Learning to prove theorems via interacting with proof assistants. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6984–6994. PMLR, 2019.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R. J., and Anandkumar, A. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, 2023. Datasets and Benchmarks Track.

Zheng, K., Han, J. M., and Polu, S. MiniF2F: A cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

A. Classification Details

Table 6 gives the full tactic-family classification table behind the compact main-text comparison. The main reading is the same across models: head families are learnable from visible proof-state text, while macro-F1 remains low because

many tactic families appear only a few times. The future-premise rows are kept as a hindsight comparison rather than as deployable inputs.

Table 6. Complete RQ1 tactic-family prediction table under by-theorem splitting. This is the full version of Table 2.

Representation	Model	Accuracy	Macro-F1	Top-3
Raw text	Majority	0.262	0.007	0.495
	Keyword	0.168	0.015	0.449
	Naive Bayes	0.331	0.066	0.579
	TF-IDF LR	0.105	0.066	0.240
	TF-IDF SVM	0.221	0.099	0.460
Normalized text	Majority	0.262	0.007	0.495
	Keyword	0.168	0.015	0.449
	Naive Bayes	0.335	0.065	0.588
	TF-IDF LR	0.109	0.066	0.245
	TF-IDF SVM	0.214	0.096	0.456
Structured	Majority	0.262	0.007	0.495
	Keyword	0.168	0.015	0.449
	Naive Bayes	0.334	0.047	0.570
	TF-IDF LR	0.103	0.059	0.219
	TF-IDF SVM	0.161	0.078	0.320
Future premise	Majority	0.262	0.007	0.495
	Keyword	0.168	0.015	0.449
	Naive Bayes	0.360	0.051	0.586
	TF-IDF LR	0.130	0.076	0.288
	TF-IDF SVM	0.296	0.124	0.546

B. Execution Details

The tables below give the details behind the Lean checking experiment in RQ3. Table 7 gives the paired comparison against unguided retrieval. Table 8 separates exact trace recovery from Lean acceptance, showing how often Lean accepts a different top-five tactic. Table 9 separates reconstruction failures from ordinary Lean rejections.

The main comparison counts every sampled state. This matches what a search system would experience: a tactic that is not reconstructed successfully for Lean checking provides no verifier feedback. Candidate-level rates over successfully reconstructed items are reported only for interpreting reconstruction quality.

Table 7. Paired Lean checking comparison against unguided retrieval on Accept@5. Differences are strategy minus unguided over the same 500 states.

Strategy	ΔA@5	95% CI	McNemar p
Soft family	-0.020	[-0.042, 0.002]	0.121
Hard family	-0.092	[-0.132, -0.052]	< 0.001
Family RRF	-0.050	[-0.080, -0.020]	0.002
Top-m family	-0.082	[-0.118, -0.046]	< 0.001

C. S4-E Extension

S4-E appends two additional dense-core modules. Classification metrics are six-seed by-theorem means with population standard deviations; retrieval metrics are exact-tactic success at rank 1 on the corresponding main split.

The extension checks whether the same pattern survives after adding two more dense-core modules: state-visible features still carry family signal, future-premise inputs remain a hindsight comparison, and soft family guidance is best read as a mild ranking preference rather than a reliable Lean-acceptance improvement.

Table 8. Lean-accepted alternatives that differ from the traced tactic. All rates use the same query-level denominator as the 500-state execution set. A-no-E is acceptance without a trace hit; E-no-A is a trace hit without Lean acceptance in the reconstructed state.

Strategy	Trace	Lean	Non-trace	A-no-E	E-no-A
Unguided	0.168	0.672	0.628	0.534	0.030
Soft family	0.170	0.652	0.596	0.516	0.034
Hard family	0.090	0.580	0.518	0.502	0.012
Family RRF	0.160	0.622	0.570	0.492	0.030
Top-m family	0.124	0.590	0.564	0.490	0.024

Table 9. Reconstruction and checking failures in the 500-state Lean check. The 245 reconstruction failures are separated from normal Lean rejections; timeouts and other tool failures did not occur.

Class	Count	Cand. frac.
Unknown identifier	198	0.016
Elab./type class	47	0.004
Parse error	0	0.000
Timeout	0	0.000
Other tool failure	0	0.000

Table 10. S4-E comparison. The extension preserves the classification pattern while leaving soft-guided retrieval tied with unguided retrieval.

Metric	S4-main	S4-main + E
Steps	3723	4464
Theorems	1702	1979
Labels	81	90
NB accuracy	0.352 ± 0.004	0.357 ± 0.010
SVM macro-F1	0.129 ± 0.008	0.149 ± 0.017
Unguided E@1	0.120	0.141
Soft E@1	0.123	0.141
Hard E@1	0.069	0.075

D. Practical Design Notes

Keep the checker close to the ranker. Offline trace matches are useful for fast iteration, but the ranked list has to be sampled under Lean checking before small differences are trusted. A small reordering can leave the traced tactic unchanged while changing which alternative tactics Lean sees.

Use family predictions as nudges. Tactic families summarize the first choice a prover makes: simplify, rewrite, introduce, split, apply or close. They are dangerous as discrete routers. A rewrite-looking state may still close by `simpa`; a construction-looking state may be finished by `exact`; and arithmetic automation can replace several local moves.

Make premise retrieval part of the input story. Premise information is useful only when it is obtained before the target tactic is known. Retrieved premises are therefore part of the prover's view, while tactic-annotated premises belong to hindsight comparison. The engineering target is to improve retrieval first, then spend model capacity on tactic arguments and local names.

Read accepted alternatives as search fuel. Accepted non-trace tactics are not proof completions, but they give an agent legal branches, Lean errors and chances to learn local repairs. A method that exposes several acceptable first moves may be more useful than one that ranks a single traced move higher while hiding nearby alternatives.