Representing Lean Proofs as Trajectories in Latent Space

Elisaveta Samoylov and Soroush Vosoughi
Department of Computer Science, Dartmouth College
{lisa.v.samoylov.26, Soroush.Vosoughi}@dartmouth.edu

Abstract

Lean proofs are built as sequences of tactic-induced state transitions, yet learned models often represent proof steps primarily through tactic strings or raw proof-state text. Building on Delta Tokens, which encode a proof step by the local edit it induces between successive proof states, we train an encoder-only Transformer to learn contextualized representations of Lean proof steps from state changes. We then use these step representations to study complete proofs as trajectories in a learned latent space.

We first show that the Delta-based Transformer yields better held-out next-tactic retrieval than a matched surface-syntax control, supporting the representational choice used in the trajectory analysis. We then analyze proof trajectories using path length, endpoint span, directness, curvature, and torsion. Across the LeanWorkbook slice used here, longer proofs become increasingly indirect within a relatively bounded latent span: path length grows sharply with proof length while endpoint span changes little, mean step size decreases, curvature rises modestly, and torsion falls. Qualitative case studies show that these geometric patterns align with recognizable proof organizations, including immediate closure, aligned accumulation, scaffolded enrichment, bookkeeping-heavy restructuring, and repeated local contradiction work.

The dataset is small and heavily skewed toward short proofs, so the claims are necessarily limited. Within those limits, the results suggest that learned state-change representations recover nontrivial structure in how proofs unfold and provide a promising basis for future trajectory-aware theorem proving.

1 Introduction

Interactive theorem provers such as Lean4 construct proofs as sequences of tactics that transform one proof state into the next (de Moura and Ullrich, 2021). But the operative object at each step is not just the tactic string. It is the local transformation from the current proof state to the next one. Two syntactically different tactics can induce similar proof-state changes, while the same tactic can play quite different roles depending on context. Representations that emphasize tactic surface form or raw proof-state text therefore risk obscuring the transition structure that organizes proofs.

Samoylov and Vosoughi (2025) addressed this issue with Delta Tokens, which represent a proof step by the symbolic edit it induces between successive proof states. Under a static co-occurrence objective, that effect-based view yielded more useful step embeddings than surface-only baselines. The present paper extends that line of work in two ways. First, it replaces the static Word2Vec-style objective with an encoder-only Transformer trained on Delta Token contexts. Second, it uses the resulting proof-step representations to study complete Lean proofs as trajectories in a learned space.

This paper is therefore not primarily a new theorem prover. It is a descriptive study of proof organization in a learned transition space. Once proof steps are embedded as vectors, a full proof can be viewed as an ordered path through that space. That makes it possible to ask structural questions that ordinary retrieval scores do not answer. Do longer proofs mainly span a larger region, or do they take more circuitous routes through a similar region? Do some proofs proceed almost directly from setup to closure, while others repeatedly normalize, enrich, split, or restructure the local state before a terminal step becomes available?

These questions matter because a proof is not just a bag of steps. It has order and internal organization. Some proofs move quickly from setup to discharge. Others accumulate a family of aligned intermediate facts. Others perform repeated local reorganization before closure becomes possible. If a learned proof-step space is coherent, those differences should appear in the geometry of proof

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop), pages 1190–1202

trajectories.

In this paper, each proof becomes an ordered sequence of learned step vectors, and we analyze that sequence using path length, endpoint span, directness, curvature, and torsion. These quantities do not provide a semantic theory of proof progress. They characterize the geometry of proof-step trajectories in the learned representation. What they reveal is whether a proof moves relatively directly from setup to closure or reaches closure only after repeated local reorganization. That makes the trajectories interesting descriptively, and it also suggests a practical direction. A prover that can recognize whether a partial proof is following a direct path, an enrichment path, or a bookkeeping path may be in a better position to choose what kind of step should come next.

The contributions of the paper are as follows:

1. We train a Transformer over Delta Token step contexts and show improved held-out next-tactic retrieval over a matched surface-syntax control.
2. We introduce a trajectory view of full Lean proofs in a learned proof-step representation space.
3. We show that in this space, longer proofs become increasingly indirect primarily through larger path length rather than larger endpoint span.
4. We provide case studies showing that these geometric patterns align with recognizable proof organizations such as immediate closure, aligned accumulation, scaffolded enrichment, bookkeeping-heavy restructuring, and repeated contradiction work.

2 Related Work

Neural theorem proving has long combined symbolic proof systems with learned guidance. Earlier work used neural models to guide proof search directly (Loos et al., 2017), while later language-model-based approaches treated theorem proving as sequence generation or structured search over proof actions (Polu and Sutskever, 2020). More recent systems have scaled these ideas substantially, combining retrieval, search, decomposition, and verifier feedback.

Within Lean specifically, LeanDojo made the ecosystem substantially easier to study by exposing a programmable environment, reusable benchmarks, and retrieval-augmented theorem proving pipelines (Yang et al., 2023). At the model level, current systems such as DeepSeek-Prover-V2 and Goedel-Prover-V2 emphasize large-scale synthesis, self-correction, subgoal decomposition, and verifier-guided search (Ren et al., 2025; Lin et al., 2026). These systems are primarily evaluated by end-to-end proving performance. For broader context on deep learning approaches to theorem proving, see Li et al. (2024).

The present paper addresses a different problem. Rather than asking how to generate complete proofs, it asks how to represent proof steps and what complete proofs look like in the resulting learned space. In that respect, the closest point of departure is Samoylov and Vosoughi (2025), who argued that a proof step is better modeled by the edit it induces on the proof state than by tactic surface form alone. We keep that representation, replace the static co-occurrence objective with transformer-based masked language modeling, and then study the geometry of the resulting proof-step trajectories.

This also distinguishes the present work from sequence-only tactic models. A sequence model can succeed at next-step prediction while still leaving the latent space itself uninterpreted. Here the learned space is one of the main objects of study. The paper is also complementary to work that uses higher-level structure or proof sketches to guide formal reasoning (Jiang et al., 2023). Our contribution is not a sketching mechanism or a new search policy, but a descriptive account of how full proofs unfold in a learned space of local state-change representations.

3 Background

3.1 Lean Proof States and Step Triples

A Lean proof proceeds by repeatedly applying tactics to a proof state. At each step, the state contains a local context and one or more goals. We write a proof as a sequence of triples

$$(s_0, t_1, s_1), (s_1, t_2, s_2), \ldots, (s_{S-1}, t_S, s_S),$$

where tactic t_i transforms state s_{i-1} into state s_i. Table 1 exhibits a sample Lean proof, along with the local context states.

We use the stepwise LeanWorkbook corpus (Ying et al., 2024), which provides before–tactic–after triples directly. In the slice used here, the data contain 13,517 proofs and 25,214 proof steps. The distribution is extremely skewed toward short proofs: 58.3% of proofs contain a single tactic.

State-Before-Tactic	Tactic	State-After-Tactic
$\vdash (1-\sqrt{5})/2 < 0$	`norm_num div_neg_iff, sub_neg`	$\vdash 1 < \sqrt{5}$
$\vdash 1 < \sqrt{5}$	`exact Real.lt_sqrt_of_sq_lt (by norm_num)`	`no goals`

Table 1: Sample proof from the Lean Workbook dataset, with proof id `lean_workbook_plus_55127`. The two tactics `"norm_num [div_neg_iff, sub_neg]"` and `"exact Real.lt_sqrt_of_sq_lt (by norm_num)"` form a Lean proof.

3.2 Delta Tokens

Delta Tokens, introduced by Samoylov and Vosoughi (2025), represent a proof step by the edit from s_{i-1} to s_i. The construction has two parts. First, it records token-level additions and deletions between the before and after states. Second, it augments those lexical edits with typed structural markers such as hypothesis additions, goal closure, goal-count changes, relation-direction flips, and operator-count changes. The resulting token multiset is the Delta representation of the step.

The underlying idea is that a proof step is often better characterized by the local proof-state change it produces than by the tactic string alone. That is the representational hypothesis carried into the Transformer setup. Appendix A gives one worked example of the Delta Token construction for a single Lean proof step.

4 Methodology

4.1 Dataset and Proof-Level Splits

Each example is a proof step consisting of a triple $(s_{\text{before}}, \text{tactic}, s_{\text{after}})$. Steps are grouped by `proof_id`, and 80/10/10 train/validation/test splits are performed at the proof level so that near-identical steps from the same proof do not cross split boundaries.

Each proof step yields a short token sequence. That sequence is used for masked-language-model training and, after training, for embedding the step as a vector. By concatenating the embedded steps in proof order, we obtain a vector-valued proof trajectory.

Figure 1 makes the imbalance visible. The higher-order parts of the geometry are therefore computed on much smaller subsets than the full corpus. That does not prevent the analysis, but it fixes its scope from the start.

Statistic	Value
Number of proofs	13,517
Number of tactic steps	25,214
Average proof length	1.865
Median proof length	1.000
Single-tactic proofs (%)	58.312
Unique tactics	13,537

Table 2: Dataset statistics for the LeanWorkbook slice used in this paper.

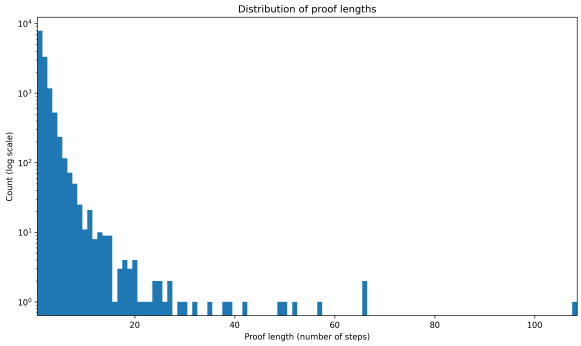


Figure 1: Distribution of proof lengths in LeanWorkbook, measured in number of proof steps. The dataset is strongly skewed toward short proofs, with most of the mass concentrated at one- and two-step proofs and a relatively thin long tail. This matters for the trajectory analysis because higher-order quantities are only defined once a proof has enough steps.

4.2 Tokenization and Anonymization

Proof states are tokenized by splitting around a fixed set of Lean-specific and mathematical operator characters, including symbols such as $\vdash$, $\leq$, $\geq$, `^`, `*`, `+`, `-`, parentheses, colons, and equality signs.

To reduce spurious correlations from local naming conventions, variables and hypotheses are anonymized deterministically within each state. Common variables such as `a`, `b`, and `c` are mapped to placeholders such as `_x1`, `_x2`, and `_x3`. Hypothesis identifiers are mapped to placeholders such as `_h1`, `_h2`, and so on. When a proof step closes all goals and the turnstile disappears, we append a dedicated marker token `NO_GOALS` so that goal closure is observable at the token level.

4.3 Delta-Token Construction

For each proof step, we construct Delta Tokens from the before and after states.

Lexical delta tokens. For each surface token w, we compute its net count change between s_{before} and s_{after}. Each net-new occurrence contributes one token `TOK_`w. This yields a multiset-like represen-

tation of what the proof step introduced.

Typed delta tokens. We augment the lexical deltas with structured edit indicators that capture higher-level proof-state changes, including hypothesis additions and removals, goal closure, changes in goal count, relation-direction flips in the goal region, and operator-count changes for a small set of symbols. If a step yields no deltas under a given ablation, an Δ_EMPTY token is inserted.

Each training example is then a short token sequence consisting of the Delta Tokens, the typed delta tokens, and a tactic identifier token TACTIC_<name>, preceded by a special [CLS] token. Padding uses [PAD].

Appendix A gives a concrete example of how one proof step is converted into lexical Delta Tokens and typed edit indicators.

4.4 Transformer Training

The original Lean2Vec setup used static co-occurrence objectives. Here we replace that with masked language modeling in an encoder-only Transformer (Devlin et al., 2019).

A fixed proportion of non-special tokens (15%) is selected for prediction. Following the standard BERT masking rule, selected positions are replaced by [MASK] 80% of the time, by a random vocabulary token 10% of the time, and are left unchanged 10% of the time. The model is trained with cross-entropy loss to recover the original token identities at masked positions.

The Transformer is a compact encoder-only model with 4 layers, hidden size 256, 4 attention heads, feed-forward dimension 512, and maximum sequence length 64. Inputs are the sum of learned token embeddings and learned positional embeddings. The final encoder states are normalized and passed to a masked-language-model projection whose weight matrix is tied to the input embedding table. Training uses AdamW with gradient clipping and a warmup-plus-decay learning-rate schedule initialized with RANDOM_SEED = 9000. Model selection tracks held-out masked-token loss and masked-token accuracy on the validation split.

4.5 What the Latent Space Represents

The Transformer directly learns a token embedding table. Every vocabulary item—lexical Delta Tokens, typed Delta Tokens, and TACTIC_* tokens—receives a learned vector. The downstream latent vectors used in the experiments are constructed from that token space.

For the Delta model, a proof step is embedded by mean-pooling the tokens produced by the edit from s_{before} to s_{after}, together with the typed edit indicators. For the surface-syntax control, a proof step is embedded by mean-pooling the tokenized state_before sequence. In both cases, the step representation is built from local proof information rather than from a standalone tactic embedding.

Tactic identity still enters training. The masked-language-model sequences include a TACTIC_* token, so the model learns to associate tactic identifiers with characteristic edit-token configurations. But in the default downstream analysis, the step vector itself is built from the context tokens rather than by directly pooling the tactic token. The latent space analyzed in the rest of the paper should therefore be read as a space of learned proof-step representations. In the Delta setting, it is specifically a space of local proof-state transitions as represented by the model.

4.6 Surface-Syntax Control

Alongside Delta Tokens, we train a matched surface-syntax control. In this setting, the input sequence is derived directly from tokenized state_before, followed by a separator and a tactic token. This gives a counterfactual condition in which the model must infer step structure from raw state text rather than from explicit state edits. The architecture, masking rule, optimizer, and evaluation protocol are otherwise unchanged.

4.7 Proof Trajectories

After training, each proof step is embedded as a vector in $\mathbb{R}^d$. If a proof consists of steps

$$(s_0, t_1, s_1), (s_1, t_2, s_2), \ldots, (s_{S-1}, t_S, s_S),$$

its trajectory is the ordered sequence of step embeddings

$$z_1, z_2, \ldots, z_S \in \mathbb{R}^d,$$

where z_i is the embedded representation of the i-th proof step.

In the Delta setting, each z_i is a pooled representation of the symbolic edit from s_{i-1} to s_i together with typed edit indicators. The trajectory therefore lives in a learned space of local proof-state transitions. In the surface-syntax control, each z_i is a pooled representation of tokenized state_before, so the trajectory is closer to a surface proof-state space.

4.8 Geometric Measurements

Given a proof trajectory $z_1, \ldots, z_S$, define step displacements

$$d_i = z_{i+1} - z_i, \qquad i = 1, \ldots, S-1.$$

We compute the following quantities.

Path length. $L = \sum_{i=1}^{S-1} \|d_i\|_2$
This is the total amount of movement along the path. If two proofs occupy a similar endpoint span but one proof accumulates more local transition changes on the way, its path length will be larger.

Mean step size. $\bar{L} = \frac{1}{S-1} \sum_{i=1}^{S-1} \|d_i\|_2$
This is defined for proofs with at least two steps. This separates proof length from the typical size of an individual local transition.

Endpoint span. $D = \|z_S - z_1\|_2$
This is the straight-line distance between the first and last step representations. It captures the net span of the trajectory independently of how indirect the path is in between.

Displacement ratio. $R = \frac{D}{L}$.
Values closer to 1 indicate straighter trajectories. Small values indicate that the proof accumulates many local transition changes that are not well aligned with the overall span.

Curvature. For proofs with at least three steps, we define local turning angles

$$\theta_i = \arccos\left(\frac{d_i^\top d_{i+1}}{\|d_i\|_2 \, \|d_{i+1}\|_2} \right), \; i = 1, \ldots, S-2,$$

and report mean curvature $\kappa = \frac{1}{S-2} \sum_{i=1}^{S-2} \theta_i$.

Curvature measures how often the form of the local transition changes from one step to the next.

Discrete torsion. For proofs with at least four steps, let d_i, d_{i+1}, and d_{i+2} be three consecutive displacement vectors. After Gram–Schmidt orthonormalization of (d_i, d_{i+1}, d_{i+2}), let u_i be the unit vector orthogonal to the plane spanned by d_i and d_{i+1}. We define local torsion as $\tau_i = \frac{|d_{i+2}^\top u_i|}{\|d_{i+2}\|_2}$, and report mean torsion $\tau = \frac{1}{S-3} \sum_{i=1}^{S-3} \tau_i$.

If curvature measures turning within a local plane, torsion measures whether the proof shifts into a new local mode of change rather than continuing to vary the same one.

All geometric quantities are reported only on proofs for which they are defined.

Representation	Acc.	R@5	MRR
Surface-syntax Transformer	0.099	0.134	0.103
Delta Transformer	**0.139**	**0.209**	**0.155**

Table 3: Held-out next-tactic retrieval for the two Transformer variants. Delta Tokens improve over the matched surface-syntax control on all three metrics. Note that the Lean Workbook dataset contains 13,537 unique tactics, so random tactic selection will pick the right tactic with accuracy < 0.001.

5 Experiments

The empirical study has two parts. The first checks that the Delta-based Transformer yields better local proof-step representations than a matched surface-syntax control. The second studies the geometry of complete proof trajectories in the resulting learned space.

For local embedding quality, we evaluate held-out next-tactic retrieval using top-1 accuracy, Recall@5, and mean reciprocal rank. For geometry, we compute all path statistics from saved proof trajectories built from the step representations described above.

6 Results

6.1 Local Embedding Quality

Local embedding quality is assessed through retrieval tasks: accuracy (Acc.), Mean Reciprocal Rank (MRR), and Recall@5 (R@5). The Lean Workbook is split 80/10/10 train/validation/test by proof, and each proof step is embedded as in Section 4.5. Accuracy assesses how well the models predict the held-out tactic from the closest training step to the test step in embedding space, where the prediction is the tactic identifier of that training step. For R@5 and MRR, training step vectors are averaged so there is one vector per tactic. These vectors are ranked by similarity to each test step, and the rank of the held-out tactic is recorded.

Table 3 gives the local retrieval result. The Delta Transformer improves over the surface-syntax control on every metric. That result matters here mainly because it supports the representational choice used in the trajectory analysis. The main contribution of the paper is not another benchmark comparison. It is what the learned proof-step space makes visible once full proofs are treated as paths.

6.2 Qualitative Structure

Figure A2 shows 2D UMAP projections of sampled proof trajectories under the two Transformer representations (McInnes et al., 2020). These visualizations are qualitative only, but they are consistent with the quantitative result. The Delta-based trajectories appear more structured and less diffuse than the surface-syntax control.

6.3 Geometry of Transformer–Delta Proof Trajectories

The geometry is only partially observable because the dataset is dominated by short proofs. Out of 13,517 proofs, 5,635 contain at least two steps and therefore admit path-based measurements, 2,307 admit curvature, and 1,129 admit torsion. We report each quantity only on the subset of proofs for which it is defined.

Table 4 gives aggregate statistics. The first thing to notice is the separation between local scale and global path complexity. Mean step size is fairly concentrated ($\bar{L} = 1.206 \pm 0.223$), and endpoint span is even tighter ($D = 1.315 \pm 0.178$). By contrast, total path length is much more variable ($L = 2.287 \pm 2.772$). The learned space does not look random. Proofs occupy a relatively bounded region, but they reach that region through paths of very different complexity.

Metric	Eligible n	Mean	Median	SD
Path length L	5635	2.287	1.443	2.772
Mean step size $\bar{L}$	5635	1.206	1.265	0.223
Endpoint span D	5635	1.315	1.339	0.178
Displacement ratio R	5635	0.788	1.000	0.282
Curvature κ	2307	1.955	1.918	0.244
Torsion τ	1129	0.801	0.842	0.163

Table 4: Geometry statistics from the Transformer–Delta proof-step trajectories. Path-based quantities require at least two steps. Curvature requires at least three. Torsion requires at least four.

The proof-length breakdown in Table 5 makes the main pattern sharper. As proof length increases, total path length rises strongly, from 1.301 in two-step proofs to 7.070 in proofs of length 5+, while endpoint span remains nearly unchanged, from 1.301 to 1.344. The main geometric change is therefore not expansion in endpoint span. It is increasing indirectness.

The displacement ratio drops from 1.000 in the two-step case to 0.561 for proofs of length 3–4 and to 0.264 for proofs of length 5+. Longer proofs do not simply move farther through the learned space. They accumulate more local transition changes inside a similar endpoint span.

Mean step size declines with proof length, from 1.301 in two-step proofs to 0.993 in the 5+ bucket. In the current data, longer proofs are composed not only of more steps, but of smaller ones. Curvature rises modestly with proof length, from 1.928 to 2.033, while torsion falls from 0.857 to 0.753. A reasonable reading is that longer proofs change direction more often, but increasingly within a more locally planar subspace of represented proof-state transitions.

Steps	n	L	$\bar{L}$	D	R	κ	τ
2	3328	1.301	1.301	1.301	1.000	–	–
3–4	1701	2.510	1.093	1.333	0.561	1.928	0.857
5+	606	7.070	0.993	1.344	0.264	2.033	0.753

Table 5: Geometry by proof-length bucket from the Transformer–Delta step trajectories. Curvature is defined only for proofs with at least three steps, and torsion only for proofs with at least four. The 3–4 torsion value is therefore computed from four-step proofs only.

Under this interpretation, the trajectories tell us about the organization of local proof-state changes as represented by the model. Straight paths correspond to proofs whose successive local edits are well aligned. More curved and indirect paths correspond to proofs that normalize, enrich, split, or rearrange the local state before they can close. The geometric result is therefore not that the space makes long proofs look geodesic. It is that the space separates relatively direct and relatively circuitous patterns of local proof-state transition.

6.4 Case Studies

The aggregate pattern is useful, but it is easier to see what the trajectories are measuring in individual proofs. Table 6 collects five representative examples from LeanWorkbook, and Figure 2 shows their paths in a shared 2D UMAP view of the learned step space. The point of these examples is not to treat geometry as a substitute for proof reading. It is to show that the path statistics line up with recognizable proof organizations.

Two-step closure. Consider the proof `rintro ... → nlinarith [sq_nonneg (a - b), sq_nonneg (b - c), sq_nonneg (c - a)]`. Its trajectory has $L = D = 1.572$ and $R = 1.0$. With only one segment, there is no internal redirection. Under the present interpretation, this is a proof with

Pattern	Abbreviated proof script	L	D	R	κ	τ
Two-step closure	`rintro ... → nlinarith [...]`	1.572	1.572	1.000	–	–
Aligned accumulation	`have := sq_nonneg ... → have := sq_nonneg ... → have := sq_nonneg ... → nlinarith [...]`	1.585	1.368	0.863	1.842	0.994
Scaffolded enrichment	`norm_num [...] → have h1 := sq_nonneg ... → have h2 := sq_nonneg ... → nlinarith`	2.731	0.700	0.256	1.649	0.514
Bookkeeping-heavy restructuring	`simp [hn] → induction' ... → norm_num → cases ... → nlinarith`	5.271	0.635	0.120	2.606	0.569
Repeated contradiction work	`contrapose! ... → have h3 := ... → simp [...] at h3 → have h4 := ... → simp [...] at h4 → omega`	6.257	1.015	0.162	2.539	0.358

Table 6: Representative proof trajectories from the Transformer–Delta model. Here L is path length, D is endpoint span, $R = D/L$ is displacement ratio, κ is mean curvature, and τ is mean torsion.

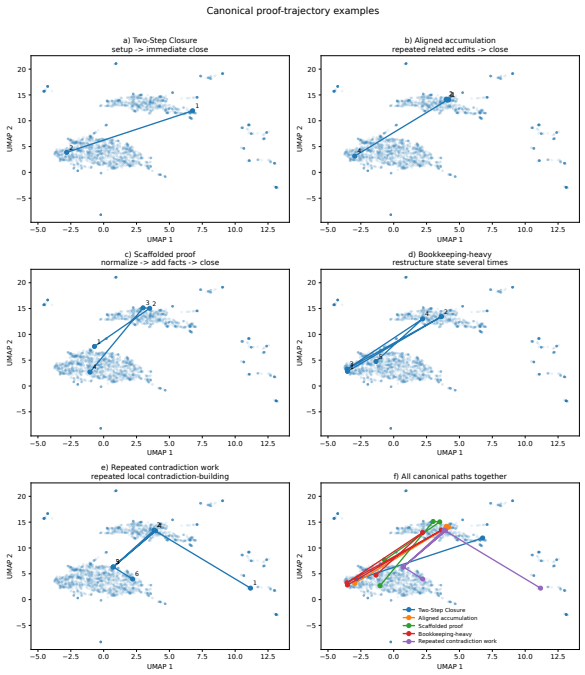


Figure 2: Canonical proof-trajectory examples projected into a shared 2D UMAP view of the learned proof-step space. Panels (a)–(e) show the five representative case-study paths individually: two-step closure, aligned accumulation, scaffolded enrichment, bookkeeping-heavy restructuring, and repeated contradiction work. Panel (f) overlays all five paths. Each trajectory is a sequence of learned proof-step representations. The two-step case is essentially a single transition from setup to discharge. The aligned-accumulation case shows repeated local edits of the same general form before a closing step. The scaffolded, bookkeeping-heavy, and repeated-contradiction cases are more indirect and move through several intermediate forms of local proof-state change before closure.

one setup transition and one solving transition. The local proof-state change introduced by the first step is already enough to make the second step terminal.

Aligned accumulation. A different four-step proof has the form `have := sq_nonneg (x^2 - y * z) → have := sq_nonneg (y^2 - z * x) → have := sq_nonneg (z^2 - x * y) → nlinarith [...]`. Here $L = 1.585$, $D = 1.368$, and $R = 0.863$. The path is longer than in the two-step case, but still fairly direct. The first three steps induce nearly the same kind of local state change. Each adds one nonnegativity fact of the same general form. The final step is different, which appears in the torsion. This is a proof with a stable preparatory phase followed by a distinct closing move.

Scaffolded enrichment. The inequality proof `norm_num [ha, hb, hc, habc, h] → have h1 := sq_nonneg (a^2 - 1) → have h2 := sq_nonneg (b^2 - c^2) → nlinarith` is more indirect. Its trajectory has $L = 2.731$, $D = 0.700$, and $R = 0.256$. The path is long relative to its span because the proof is not pushing one transition pattern all the way through. It first normalizes the arithmetic state, then enriches the context twice, and only then closes. The geometry picks up the internal architecture of the proof. It moves through normalization, enrichment, enrichment, and closure.

Bookkeeping-heavy restructuring. A more elaborate example is the proof `simp [hn] → induction' hn with n hn ih → norm_num → cases n <;> simp_all [Nat.succ_eq_add_one, pow_succ] → nlinarith`. This path has $L = 5.271$, $D = 0.635$, $R = 0.120$, $\kappa = 2.606$, and $\tau = 0.569$. Figure A3 shows the same proof in a 2D UMAP projection together with the metric definitions used in the paper. The path is long, highly curved, and narrow in end-

point span. That is exactly what the script does. It repeatedly restructures the local proof state before the final arithmetic discharge becomes available.

Repeated contradiction work. A final example is `contrapose! h_1 → have h_3 := h_2 0 7 → simp [h_1] at h_3 → have h_4 := h_2 0 0 → simp [h_1] at h_4 → omega`. This path has $L = 6.257$, $D = 1.015$, $R = 0.162$, $\kappa = 2.539$, and $\tau = 0.358$. It is long and highly curved, but the torsion remains relatively low. That means the proof keeps changing direction locally while staying inside a fairly narrow transition regime. The script itself shows the same pattern. It instantiates, simplifies, instantiates, simplifies, then closes. The proof is not introducing many different kinds of local proof-state change. It is revisiting one small family of them until contradiction becomes explicit.

Taken together, these cases suggest that the trajectories distinguish at least five proof organizations. They capture immediate closure, aligned accumulation, scaffolded enrichment, bookkeeping-heavy restructuring, and repeated local contradiction work. The geometry does not replace semantic proof analysis, but it does recover something real about how the proof state is being transformed from step to step.

7 Discussion

The main interpretive point is that the geometry lives in a space of learned proof-step representations. In the Delta model, those representations are built from symbolic state edits and typed edit indicators. The path statistics should therefore be read as statistics of local proof-state transition patterns as represented by the model. Path length measures accumulated local change. Low displacement ratio means the intermediate steps are not well aligned with the overall span. Curvature measures shifts in the form of local change. Torsion measures whether those shifts remain within one local mode or move into another.

That interpretation makes the case studies easier to read. A long, indirect trajectory does not have to mean that the proof is semantically wandering. It can mean that the proof is carrying out many local transformations before closure becomes possible, or that it is repeatedly revisiting the same narrow transition pattern. A shorter, straighter trajectory can indicate that successive local changes are aligned from the start. What the measurements recover is not theorem difficulty in the abstract, but structure in how the proof proceeds.

There is also a data issue that matters for interpretation. LeanWorkbook is dominated by short proofs, so higher-order measurements are computed on a much smaller subset than the full corpus. That does not invalidate the result, but it does bound its scope.

Within those bounds, the trajectory view is still useful. Path length, mean step size, directness, curvature, and torsion separate different aspects of proof organization. They distinguish immediate closure from aligned accumulation, scaffolded enrichment, bookkeeping-heavy restructuring, and repeated local contradiction work.

8 Trajectory-Guided Proving

The present paper is descriptive. It shows that proofs trace structured paths in a learned space of local proof-step representations, and that those paths separate several recognizable proof organizations. The next step in this line of work is to test whether that structure can be used during proof search.

The motivating idea is simple. A prover does not only need to know which next step is locally plausible. It also needs to know what kind of local change the current proof still seems to require. A partial proof may already look like it is on a direct path to closure, in which case a terminal step may be appropriate. It may instead look like a scaffolded proof whose trajectory still reflects enrichment or normalization, in which case an early closing move is less likely to succeed. It may also look like a bookkeeping-heavy proof that is still reorganizing the local state, where the better move is not a discharge step but another restructuring one.

Under that view, a trajectory can serve as an additional signal over candidate next steps. A candidate can be evaluated not only by whether it is legal or locally probable, but also by how it extends the current path. Does it preserve a transition pattern that has been productive so far? Does it introduce a new mode of local change at a point where successful proofs typically do so? Does it force the path toward closure before the proof state appears ready for it? These are trajectory questions rather than token-level questions. That use case is not claimed here as a finished result. It is where the work is leading. This paper establishes the representation

and the descriptive geometry. The natural continuation is to test whether trajectory-aware signals can improve step selection in an LLM-based Lean prover, especially in settings where several next tactics are superficially plausible but only some are consistent with the way successful proofs usually unfold. More broadly, this direction is compatible with recent work that uses proof sketches or higher-level intermediate structure to guide search (Jiang et al., 2023).

9 Conclusion

This paper studies full proofs as trajectories in a learned space of proof-step representations. In the Delta setting, each step vector is built from the local symbolic edit between consecutive proof states, so the resulting geometry should be read as a geometry of represented local proof-state transitions rather than of tactic labels alone.

In that space, longer proofs do not primarily expand in endpoint span. They become less direct. Total path length rises sharply while the straight-line span between the opening and closing step profiles changes very little. Mean step size falls with proof length, curvature rises modestly, and torsion falls. The case studies show that these aggregate patterns correspond to recognizable proof organizations like immediate closure, aligned accumulation, scaffolded enrichment, bookkeeping-heavy restructuring, and repeated contradiction work.

These are limited claims, but they are clean ones. They show that a Delta-based proof-step space supports a nontrivial path-based description of how proofs unfold. That is the main result of the paper.

Limitations

This study has several limitations.

First, the data are small and heavily skewed toward short proofs. More than half of the proofs contain only one tactic. That leaves 5,635 proofs for path-based measurements, 2,307 for curvature, and 1,129 for torsion. Any claim about long-horizon proof geometry should therefore be read as preliminary.

Second, the trajectories are defined over learned proof-step representations rather than explicit proof-state embeddings. This is exactly what makes the present analysis possible, but it also means that the geometry is an account of local transition structure in the learned representation, not a direct account of how theorem states evolve semantically.

Third, Delta Token construction requires explicit before and after states for each proof step. Many widely used theorem-proving datasets do not provide these triples directly and would require additional preprocessing.

Fourth, the current experiments are still a small study. The Transformer comparison should be rerun under multiple random seeds, and the geometric observations should be checked on larger Lean corpora, especially corpora with longer proofs.

References

Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2023. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *Preprint*, arXiv:2210.12283.

Zhaoyu Li, Jialiang Sun, Logan Murphy, Qidong Su, Zenan Li, Xian Zhang, Kaiyu Yang, and Xujie Si. 2024. A survey on deep learning for theorem proving. *Preprint*, arXiv:2404.09939.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. 2026. Goedel-Prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. In *The Fourteenth International Conference on Learning Representations*.

Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. 2017. Deep network guided proof search. *Preprint*, arXiv:1701.06972.

Leland McInnes, John Healy, and James Melville. 2020. UMAP: Uniform manifold approximation and projection for dimension reduction. *Preprint*, arXiv:1802.03426.

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

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. 2025. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *Preprint*, arXiv:2504.21801.

Elisaveta Samoylov and Soroush Vosoughi. 2025. Modeling Tactics as Operators: Effect-Grounded Representations for Lean Theorem Proving. In *Proceedings of the 3rd Workshop on Mathematical Natural Language Processing (MathNLP 2025)*, pages 168–175, Suzhou, China.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. 2023. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. Lean Workbook: A large-scale Lean problem set formalized from natural language math problems. In *Advances in Neural Information Processing Systems*, volume 37, pages 105848–105863. Curran Associates, Inc.

A Worked Delta-Token Example

This appendix gives one concrete example of how a Lean proof step is converted into a Delta Token representation. The construction follows Samoylov and Vosoughi (2025), but it is included here because the present paper relies on that representation throughout.

State before, tactic, and state after.

State-Before

a b c : $\mathbb{R}$
ha : $0 < a$ hb : $0 < b$ hc : $0 < c$
habc : $a * b * c = 1$ h : $a^4 + b^4 + c^4 = 1$
$\vdash \quad \frac{a^3}{(1-a^8)} + \frac{b^3}{(1-b^8)} + \frac{c^3}{(1-c^8)} \geq \frac{9}{8}$

Tactic

```
have h1 := sq_nonneg (a^2 - 1)
```

State-After

a b c : $\mathbb{R}$
ha : $0 < a$ hb : $0 < b$ hc : $0 < c$
habc : $a * b * c = 1$ h : $a^4 + b^4 + c^4 = 1$
h1 : $0 \leq (a^2 - 1)^2$
$\vdash \quad \frac{a^3}{(1-a^8)} + \frac{b^3}{(1-b^8)} + \frac{c^3}{(1-c^8)} \geq \frac{9}{8}$

Tokenization and anonymization. The added hypothesis

$$\texttt{h1} \ : 0 \leq (a^2 - 1)^2$$

is tokenized into symbols and identifiers, then anonymized so that local names do not leak across proofs:

$$\texttt{\_h6} \ : 0 \leq (\_x1^2 - 1)^2.$$

Token-level deltas. Let

$$\Delta(w) = \mathrm{count}_{\mathrm{after}}(w) - \mathrm{count}_{\mathrm{before}}(w).$$

In this step, the only positive deltas come from the new hypothesis. The added multiset is

$$\{\texttt{\_h6}, \texttt{:}, 0, \leq, \texttt{(}, \texttt{\_x1}, \texttt{\^{}}, 2, \texttt{\^{}}, 2, \texttt{-}, 1, \texttt{)}\}.$$

Typed edit indicators. This step also yields typed structural edits:

$$\Delta_{\texttt{ADD\_HYP}}, \qquad \Delta_{\texttt{ADD\_SYM\_-}}, \qquad \Delta_{\texttt{ADD\_SYM\_}\leq}.$$

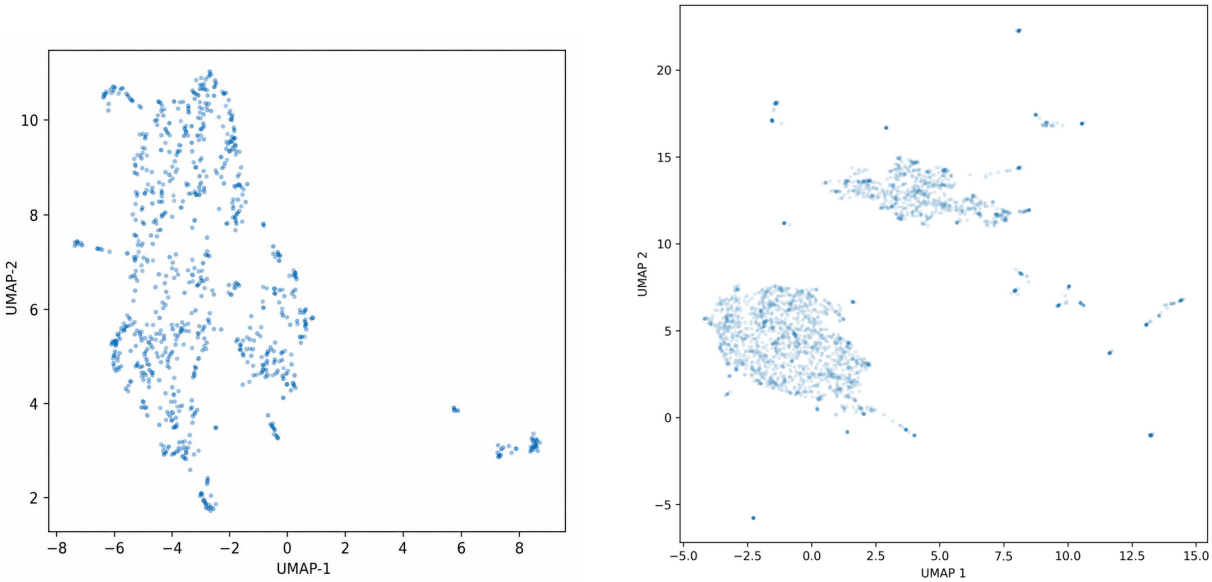
Final Delta context. The proof step is therefore represented by the union of lexical edit tokens and typed edit indicators:

$$\{\texttt{TOK\_\_h6}, \texttt{TOK\_:}, \texttt{TOK\_0}, \texttt{TOK\_}\leq, \texttt{TOK\_(}, \texttt{TOK\_\_x1}, \texttt{TOK\_\^{}}, \texttt{TOK\_2}, \texttt{TOK\_\^{}}, \texttt{TOK\_2}, \texttt{TOK\_-}, \texttt{TOK\_1}, \texttt{TOK\_)}\}$$

together with

$$\{\Delta_{\texttt{ADD\_HYP}}, \Delta_{\texttt{ADD\_SYM\_-}}, \Delta_{\texttt{ADD\_SYM\_}\leq}\}.$$

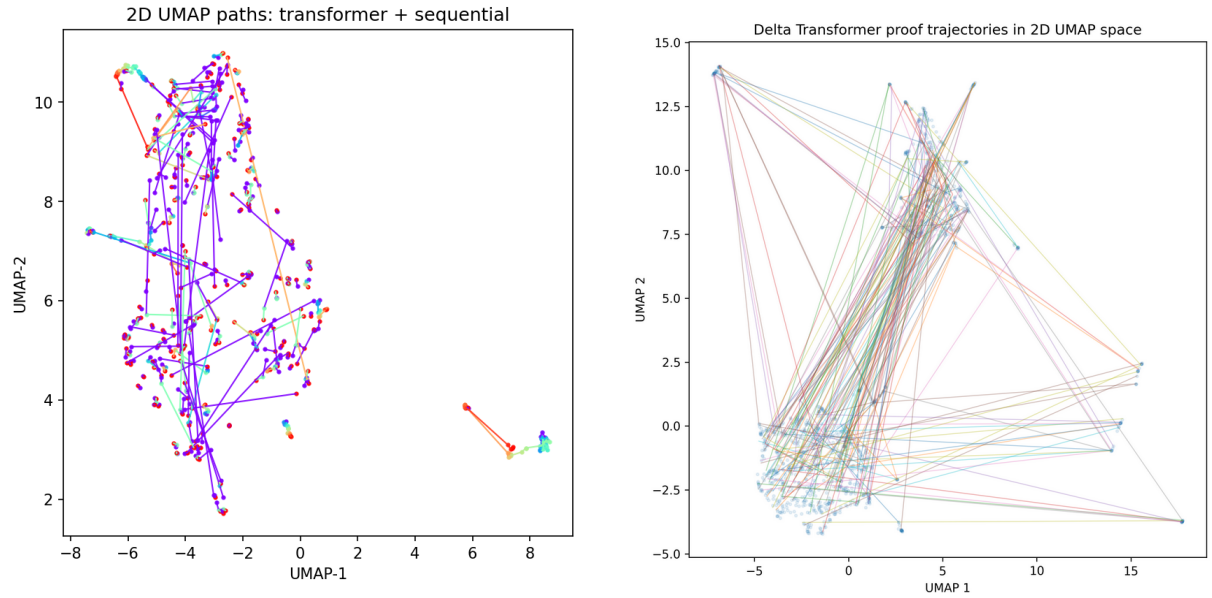
This is the kind of step representation that is embedded and then assembled into proof trajectories in the main paper.



(a) Surface-syntax Transformer step embeddings.

(b) Delta Transformer step embeddings.

Figure A1: 2D UMAP projections of step embeddings under the two Transformer representations. As in the trajectory figure, these plots are qualitative only.



(a) Surface-syntax Transformer.

(b) Delta Transformer.

Figure A2: 2D UMAP projections of sampled proof trajectories under the surface-syntax and Delta Transformer embeddings. These plots are qualitative only and are not used as evidence for the geometric claims. They simply illustrate that the Delta-based trajectories appear less diffuse.

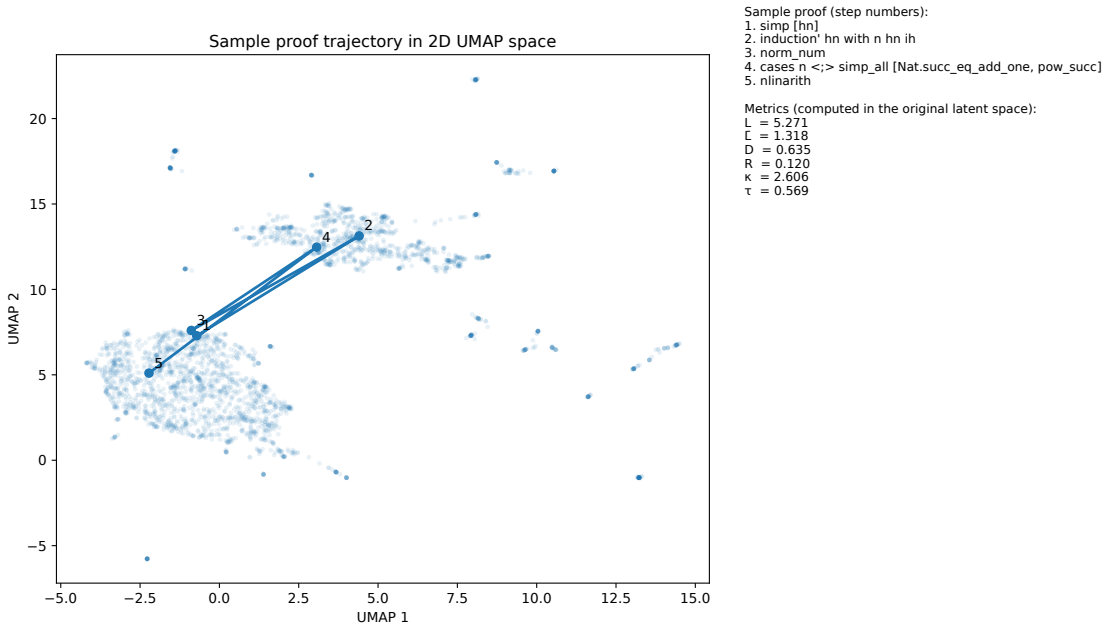


Figure A3: A sample proof trajectory in a 2D UMAP projection of the learned proof-step space. The numbered vertices correspond to the successive proof steps `simp [hn]` → `induction' hn with n hn ih` → `norm_num` → `cases n <;> simp_all [Nat.succ_eq_add_one, pow_succ]` → `nlinarith`. The metrics shown at right are computed in the original latent space, not in the 2D projection. With step displacements $d_i = z_{i+1} - z_i$, path length is $L = \sum_i \|d_i\|_2$, mean step size is $\bar{L} = \frac{1}{S-1} \sum_i \|d_i\|_2$, endpoint span is $D = \|z_S - z_1\|_2$, and displacement ratio is $R = D/L$. Curvature is the mean turning angle between successive displacements, and torsion measures the extent to which successive turns leave the local plane spanned by the previous two displacement vectors. For this proof, the long path length, small endpoint span, and low displacement ratio make the restructuring phase visible.