CS 2840: Optimal Transport and Machine Learning

ProofGOAT: Proof Generation with Optimal Transport

Name(s): Ege C., Jasmine A., Kayden K., Emmanuel R., Aadity S.
Submission Date: December 1, 2025

Abstract

We develop an Optimal Transport (OT) framework for aligning natural-language proofs with their corresponding formal Lean proofs at the token level. Prior autoformalization systems rely on contrastive objectives, retrieval heuristics or discrete search, but do not model the full geometric relationship between natural language and formal language proof manifolds. Our method learns a Neural OT transport map between token-level embedding distributions extracted from a Lean-specialized language model. To address variable-length sequences and preserve positional structure, we introduce a novel approach of constructing a void-token mechanism that converts the transport problem as balanced OT. The learned transport map is used to construct soft token prompts that guide a downstream Lean-generating decoder. Experiments on the Herald dataset demonstrate that Neural OT can produce stable bidirectional mappings. Further, analysis of token-level couplings showing meaningful structural correspondences between modalities. Our results highlight the potential for using OT-based alignment for formal proof translation.

1 Introduction

Optimal Transport (OT) theory is a mathematical framework for finding the most efficient way to move one distribution of mass to another, minimizing a total cost. Mathematically, this problem is cast as comparing two probability distributions, and considering the ways in which the distributions can be mapped to each other. In recent years, Optimal Transport Theory has found use in computing tasks such as computer graphics, image processing, and machine learning [1]. Specifically, we employ OT as a geometric alignment method in machine learning, where OT is employed to compare, match, or transform high-dimensional representations across modalities. In this work, we investigate the role of OT in the alignment of natural-language mathematical proofs and their fully formal counterparts in Lean.

Recent work has explored automated theorem proving and natural-language formalization through embedding methods and large-scale synthetic proof generation. ProofBridge [2] learns a joint NL–Lean embedding space using sequence-level contrastive training, enabling retrieval of formal proofs from natural-language prompts but without modeling token-level structure or fidelity. Complementary work such as LeanDojo [3] and ProofOptimizer [4] focus on data generation and reinforcement-based proof optimization, demonstrating that large synthetic Lean databases can drive improvements in downstream provers. However, these systems rely on alignment losses or iterative search rather than a distribution-matching mechanism such as OT. To our knowledge, no prior work applies OT to align the token-level embedding distributions of natural-language and formal proofs, or in any other manner for this downstream task.

We propose that there is a lack of trustworthy automated translation methods between synthetic reasoning data, such as Lean code, and natural language, and believe Optimal Transport methods can be helpful. Our goal is to develop a continuous, OT–based mechanism for aligning natural-language proofs with their corresponding Lean formal proofs at the token level. Existing auto-formalization systems rely on contrastive objectives, heuristic retrieval, or discrete search, but they do not model the full geometric relationship between the two modalities. We aim to learn a transport map that captures how semantic structure in natural language can be moved into the structured, type-theoretic space of Lean proofs. By incorporating Neural OT with a void-token approach for handling variable-length sequences with length unknown before the transport, our system seeks to enable principled alignment, support bidirectional translation, and ultimately improve automatic generation and verification of formal proofs. Our code can be found at https://github.com/Ege-Cakar/HERMES/.

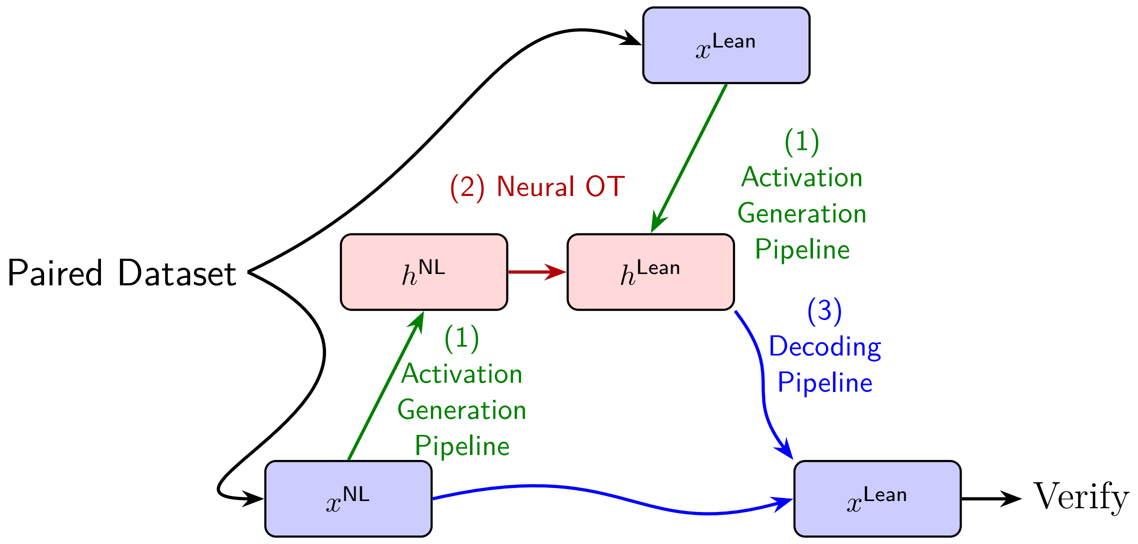

Figure 1: Overview of the Neural OT pipeline.

2 Related Work

Prior work on joint natural-language–to–formal-language representations includes *ProofBridge: Auto-Formalization of Natural Language Proofs in Lean via Joint Embeddings* [2]. Their system learns a shared Natural Language (NL) to Formal Language (FL) embedding space using contrastive learning, where matched natural-language and Lean theorem–proof pairs are close and mismatched pairs are far apart. An NL encoder and an FL encoder that linearizes Lean proof DAGs are trained jointly with a symmetric loss. This alignment enables retrieval of relevant Lean proofs from natural-language inputs, achieving strong separation between true pairs (≈ 0.64 cosine) and negatives (≈ -0.01).

ProofBridge introduces a two-stage autoformalization pipeline that converts natural-language theorem–proof pairs into Lean. First, the system performs syntactic verification, where the generated Lean theorem–proof pair is compiled using Lean's type checker; any failure produces structured diagnostic feedback indicating the exact error location and message. Second, it performs semantic verification, in which an LLM-based equivalence judge assesses whether the generated Lean theorem faithfully represents the original natural-language statement. These two signals jointly determine whether a proof is accepted or must be repaired. ProofBridge integrates these components into an iterative refinement loop, where failures trigger an LLM-driven repair step that updates the Lean theorem and proof until both syntactic and semantic criteria are satisfied or a repair budget is exhausted. We adapt this autoformalization and verification procedure in our setting, maintaining the same syntactic and semantic checks while substituting our own translation mechanism for the initial theorem–proof generation.

Work on large-scale proof-dataset creation has been advanced by *ProofOptimizer: Training Language Models to Simplify Proofs without Human Demonstrations* [4]. Using the Goedel-Prover-V2-32B to generate full Lean proofs, they were successfully able to generate 145K proof theorems, which became their training dataset. Moreover, ProofOptimizer is trained via expert iteration and reinforcement learning, using Lean to verify simplifications and provide training signal, without the need for human supervision. Their work demonstrates how a model can be trained and fine-tuned in order to convert natural language solutions into Lean proof sketches.

Beyond joint embedding and contrastive methods, recent work has explored using optimal transport directly in neural models. *Neural Optimal Transport*[5] replaces discrete OT solvers with neural networks that learn continuous transport maps, enabling scalable OT computation in high-dimensional spaces. Our approach extends this idea to the domain of proof embeddings, treating token-level translation between natural language (NL) and formal language (FL) as a continuous transport problem in the embedding space. Together, these lines of work outline the landscape of natural–formal proof alignment, large-scale synthetic proof generation, and continuous alignment methods. They provide the conceptual backdrop for approaches that aim to learn structured relationships between natural-language mathematical reasoning and its formal counterparts.

3 Method

3.1 Activation Generation

To align the NL proofs with their Lean formal proofs, we construct a paired embedding dataset from the Herald dataset [6], which consists of 45k NL-FL proofs. From the full data, we randomly subsample 5k pairs, and send the data in batches. For each sampled example, we extract the natural language proof and its aligned lean proof. We load a frozen transformer model, detailed in the following section. Hidden-state representations are extracted using a uniform procedure that tokenizes each input sequence, performs a single forward pass through the model, and returns the activations right before the last linear layer. Rather than applying mean-pooling or other projection steps, we retain the full token-level embeddings, producing variable-length matrices of shape: $(num\_tokens) \times (hidden\_dim)$. These embeddings preserve local proof structure and are suitable for distributional alignment via Neural Optimal Transport.

The final embeddings are serialized as Parquet files using batched writes. In the Parquet file, each row contains a proof identifier and its corresponding embeddings, allowing efficient streaming without loading the entire dataset into memory. The output of this pipeline is a pair of aligned embedding sets: one for natural-language proofs and one for Lean proofs. These pairs serve as the empirical distributions between which we learn a continuous Neural OT transport map.

3.2 Model

For the frozen LLM of our auto-formalization pipeline, we use Kimina-Prover-Distill-1.7B, which is a distilled version of the full-size Kimina-Prover based on Qwen3-1.7B[7]. Kimina Prover is a collaboration between Kimi and Numina, and is a 72B SOTA Open-Source Lean model[8]. Kimina-Prover-Distill-1.7B only has a hidden size of 2048, which allows for a more tractable setup, given the prevalence of the curse of dimensionality in OT [9].

3.3 Neural OT

Our formulation follows the dynamic viewpoint of Optimal Transport, where a transport map is realized by integrating a time-dependent velocity field. In the Benamou-Brenier formulation of OT [10], transport between distributions is expressed as a continuous flow governed by a velocity field $v(x,t)$ that evolves points along trajectories satisfying the continuity equation. Modern neural instantiations of this idea represent the dynamics through a neural ODE [11], parameterizing the flow as

$$\frac{dx(t)}{dt} = v_\theta(x(t), t),$$

and directly learning the vector field v_θ rather than solving a discrete OT problem. To make this approach scalable in high-dimensional embedding spaces, we adopt the flow-matching objective of [12, 13], which trains v_θ to match idealized interpolating trajectories sampled at random time points.

OT Grounding

We directly implement Benamou-Brenier dynamics at the token level, transporting empirical distributions over token embeddings. This preserves sequence structure while remaining grounded in OT theory. Unlike pooled approaches or discrete couplings, token-level continuous flows enable fine-grained analysis of how individual proof components transform between domains.

In our setting, the inputs are token embeddings originating from sequential data, so we extend the classical dynamic OT formulation by conditioning the velocity field on both the evolving token embedding and its positional structure. Concretely, we parameterize the field as:

$$v_\theta(x(t) + p + t),$$

where p is a sinusoidal positional embedding encoding token index, and $t \in [0,1]$ is a sinusoidal embedding with its own dimension projected with a linear layer afterwards to the transformer embedding dimension. We add the time embeddings rather than condition on them, a la historical Diffusion models and U-Nets[14]. Integrating this ODE from $t = 0$ to $t = 1$ defines a continuous transport map between natural-language and Lean proof embeddings. This sequence-aware Neural OT flow preserves both semantic and structural geometry and is suitable for alignment at the token level. We learn a proper flow field, which enables bidirectional translation between formal and natural proof representations. Interestingly, we see that we converge to this without a reconstruction component to our loss.

Void Tokens

To solve neural OT, we note that flow matching alone cannot create or destroy points in the trajectory space. To address this in the unbalanced OT setting, we introduce explicit void tokens corresponding to the difference $N_{\text{max}} - N_{\text{token}}$. We assign a prior to these void tokens by distributing them uniformly across the positional domain. Each missing position is filled with a void token equipped with a positional embedding. For the i-th position, the corresponding representations are defined as follows:

$$\text{void\_token}_i = [\,\text{pos\_enc}_i,\ 0\,], \text{real\_token} = [\,\text{content} + \text{pos\_enc}_i,\ 1\,].$$

This converts sequence pairs into a balanced OT problem by allowing mass preservation, enabling the velocity field to be learned consistently, even when token counts differ.

Adding positional encodings this way also enforces soft-matching regarding the positions – tokens that are earlier in the NL proof are more likely to be mapped to the tokens earlier on in the FL proof, and vice versa.

Flow Matching Objective

We seek to compute how close the model's predicted velocity field v_{theta} is to the ideal velocity field found via interpolating the OT coupling. We compute the coupling $P \in \mathbb{R}^{N_{max} \times N_{max}}$ between the NL tokens x_0 and the lean tokens x_1, using the Python Optimal Transport library [15]. This gives us a soft alignment of P_{ij} which corresponds to the probability that the NL token i maps to the lean token j.

We then compute the barycentric OT-aligned targets [16]. Thus, for each NL token i:

$$x_{1,i}^{aligned} = \sum_j P_{i,j} x_{1,j}$$

.

This yields the OT-algined target embedding for each source token. Given these targets, we generate interpolated points along straight-line geodesics by sampling $t \sim \text{Uniform}(0,1)$ and forming

$$x(t) = (1-t)x_0 + tx_1^{aligned}$$

The ideal velocity is given by the vector field:

$$v^*(x_0) = x_1^{aligned} - x_0$$

and the model predicts $v_\theta(x(t) + t)$ (abstracted away p into x for notation). The training objective is the flow-matching loss implemented as the MSE over all tokens:

$$\mathcal{L} = \mathbb{E}[\|v_0(x(t) + t) - v^*(x_0)\|^2]$$

This objective trains the velocity field to yield a continuous Neural OT map between the NL and Lean proof embeddings.

3.4 Decoding Pipeline

After learning a continous Neural OT map between the embeddings, the final step is to generate a discrete Lean proof that we can verify with the original input. For this stage, we use the Kimina-Prover-Distill-1.7B model as our decoder. We follow a soft-token prompting strategy, where we append the mapped embeddings as soft tokens to the original token-level prompt. Prior work in parameter-efficient prompt tuning demonstrates that such continuous prompts can influence model predictions in a manner comparable to lightweight fine-tuning [17, 18]. We tell the model that these soft tokens can be used as additional guidance during decoding of the target lean proof.

In a perfect mapping, the neural OT provides an almost complete solution that can be projected back into the space of token vocabulary; however, we note the novelty of the application and its respective challenge with predicting accurate embeddings. The soft token pipeline, however, is a valid method of decoding as long as the mapped results are relatively performant.

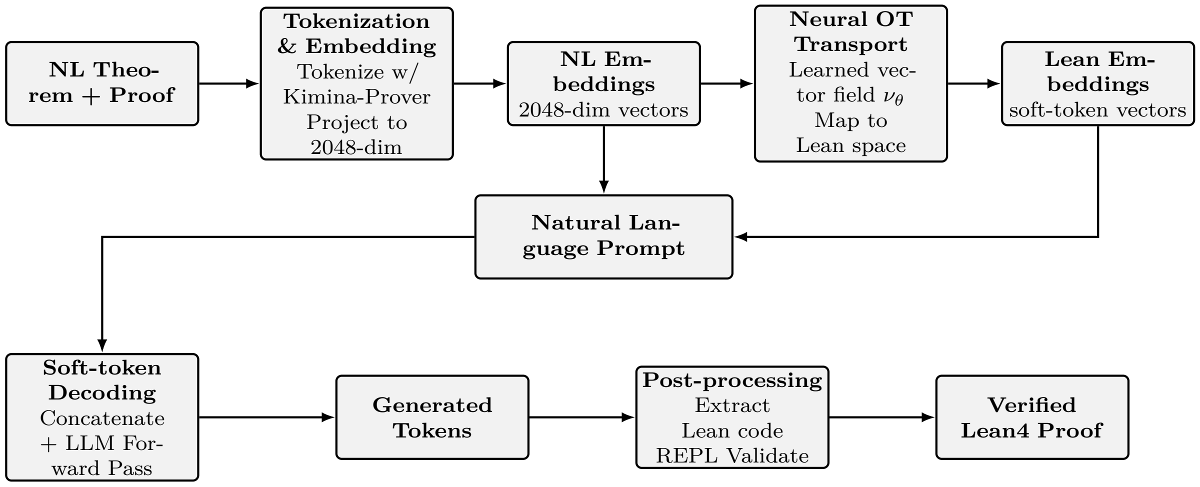

Figure 2: Overview of the End-to-End pipeline used for inference.

4 Experiments

Neural OT

The main experiments we ran for making sure the code worked via generating high-dimensional Gaussians, where learning the mappings should be relatively simple. Afterwards, we ran sweeps for finding reasonable hyperparameters; however, due to the time and computational costs of training a proficient model, this search was not exhaustive. For the architecture, we ended up going with a classic transformer encoder after trying an MLP first, and in terms of size, we ended up in the 300-310M size range. The final model was trained on an H100 for over ≈10 hours and only 5000 examples from the dataset. The version from the 7th epoch was used for final evaluations, as we were vary of overfitting with the further epochs due to the low data size. We believe it's reasonable to expect a bump in performance with more data, which we had, but didn't have time to run. However, we woudl like to note that we ran our model and calcuated armax cosine, instead of barycenter consine. We implemented multiple ways to achieve neural OT, such as concatenating time and positional embeddings, implementing initial conditioning of positional embedding, and also attempting to use different proofs from separate models to compare the exhaustiveness of the proofs we generated. We ran into many problems when solving the problem of using time and positional encodings together, to which we implemented many solutions and settled upon the solution of concatenating the two. We chose to focus on both in order to preserve the structure of the proofs for the highest mapping accuracy.

End-to-End Pipeline Engineering

The tokens generated will include both reasoning tokens in natural language and the desired lean code. To deal with this our wrapper enforces structured output with special token delimiters and also few-shot examples. The prompt is adapted from Jana et al. [2] where we also concatenate the embeddings from doing inference on our trained Neural OT model after the prompt has been tokenized and embedded to the hidden layer dimension. See Appendix C for the prompt used.

We use a standard REPL verifier for Lean4 which is open-sourced [19] and report examples of generations with and without the Lean Embeddings added as soft virtual tokens. Finer details of the whole pipeline can be found in Appendix A and examples of verified outputs in Appendix B.

5 Results and Analysis

Results for the Neural OT Model

For Natural Language to Lean translation, we achieve a mean cosine similarity score of **0.57**, which falls short of Proofbridge's score of 0.64 [2]. The Wasserstein-2 metric is equivalent to the Wasserstein-2 distance calculated between the ground truth and the results of the Neural OT model. Figure 3 shows steady optimization progress across epochs and convergence.

Table 1: Performance metrics (Mean ± Std) for translation and reconstruction tasks.

Task	**Cosine Similarity**	**Wasserstein-2**
NL to Lean	0.5740 ± 0.0251	70.7962 ± 2.7459
Lean to NL	0.5693 ± 0.0088	89.5141 ± 1.3802
Circle NL	0.9896 ± 0.0032	10.8603 ± 1.5717
Circle Lean	0.9999 ± 0.0001	0.5980 ± 0.0779

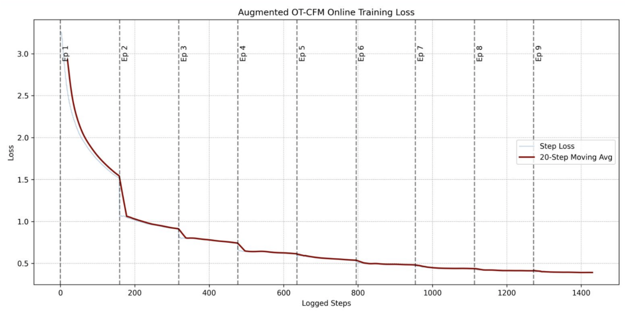

Figure 3: Training loss curve for the Neural OT model

Figure 4: Example of Token-Matching Between Lean and NL Proofs

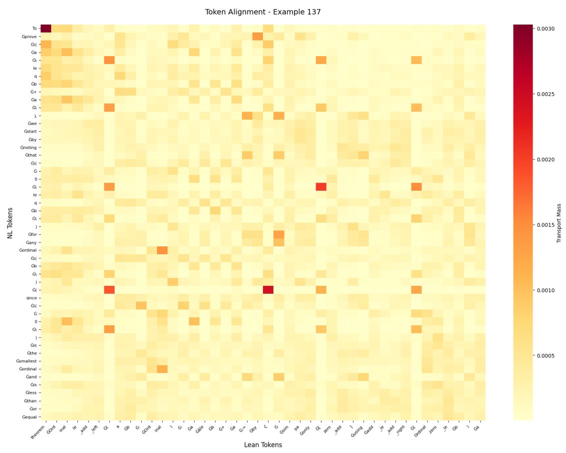

Table 2: Top 10 Token Matchings as Seen in Heatmap

NL Token	Lean Token	Mass
To	theorem	0.00302938
(	\n	0.00242564
\	[	0.00202776
(	(	0.00184703
\	(	0.0014925
\\	(	0.00145486
ordinal	inal	0.00144573
\	(	0.00136739
\	(	0.00134643
prove	by	0.00134578

Token-Level Transport Analysis

Figure 4 illustrates the coupling matrix for an example proof, and Table 2 summarizes the ten highest-mass correspondences. From the table, we note that the transport primarily assigns significant mass to structural tokens such as parentheses, backslashes, and newline markers. These tokens appear frequently in both modalities and therefore dominate early-stage alignment. Several correspondences arise from byte-pair segmentation (e.g., *ordinal* → *inal*), reflecting subword overlap. Only limited semantic alignment is visible at this stage–for example, *prove* aligning with *by*, which corresponds to common inferential patterns in both natural-language explanations and Lean tactic blocks. Overall, the learned transport map captures consistent syntactic structure, while deeper semantic alignment remains weak under the current token-level OT formulation. Our goal of preserving proof structure was achieved by our use of time and positional embeddings.

Qualitative analysis of pipeline output

Appendix B presents a comparison of generation quality for select examples, contrasting lean code produced without soft tokens (baseline) against code generated with them (our method). Currently, there is no LLM response-evaluation server robust enough for Lean, so we primarily rely on the provided REPL server for assessment. Direct comparison between the baseline (no soft tokens) and our method can be challenging, as in many cases one method produces compilable Lean code while the other does not. In a number of examples, the baseline model tends to embed more reasoning tokens directly within the Lean code, whereas our method can leverage soft tokens to copy relevant logic during decoding. A common failure mode across both approaches is the duplication of Lean expressions, particularly repeated import statements at the beginning of the file. Although such outputs may still compile, they are semantically meaningless. The baseline output in the first example of the appendix illustrates this anti-pattern clearly. Further analysis is required to determine whether the generated proofs genuinely establish the intended theorems. We generally see higher matchings towards general non-proof-related terms such as the terms "we", "proven", etc, that have more to do with jargon and less with the formal proofs themselves.

6 Discussion

Neural OT score computation

There was a subtle but important oversight in how the cosine similarity scores were computed. Specifically, the cosine similarity metrics were calculated against the argmax of the transported vectors, while the neural-OT model is actually trained to predict the barycenter of each paired distribution. Because the argmax and barycenter represent different target points, the reported cosine similarities do not accurately reflect the model's true performance. As a result, the scores above are misleading and very likely underestimate the real model accuracy, since the comparison was made against a mismatched target. We intend to have correct values in our poster. Moreover, it's crucial to note that it is impossible to get a perfect score / a score close to 0 with the W-2 metric here, as it's being calculated against the soft pairing, whereas the model can only do hard mappings – i.e transport vectors.

Limitations

A key limitation of token-level Optimal Transport (OT) is that it treats proofs as unordered empirical distributions, ignoring the graph structure of formal terms and the structure of natural-language arguments (i.e. Lean tactics). As a result, the transport map can align local geometric neighborhoods while failing to capture logical dependencies. This limits the ability of OT alone to produce coherent lean proofs without downstream support. Yet, this can be addressed by considering an iterative repair step, outlined in *ProofBridge* [2], to integrate the logical patterns of Formal Proof Language. Our own version of assuring quality-control with blank tokens comes from pulling in the nearest blank tokens from near the populated token.

End-to-end pipeline difficulties

A practical difficulty arises from our usage of approximate embeddings produced by our Neural OT model, rather than exact lean embeddings to map our soft tokens. Prior work on continuous prompts shows that soft-token conditioning is highly sensitive to the embedding initialization. When prompt vectors deviate from the pretrained embedding manifold, the model's generation process becomes unstable. Specifically, Lester et. al. [20] show that approximate or off-manifold soft tokens demonstrate a significant degradation in downstream performance. This observation aligns with broader findings that LLMs are known to be highly sensitive to perturbations in embedding space [21]. This mirrors our setting, where approximate transported embeddings can degrade performance relative to exact-embedding conditions. This sensitivity highlights the need for more robust manifold-preserving transport methods in future work.

While the OT component introduces a potential source of error in the overall pipeline, it simultaneously presents a promising avenue for future efficiency gains. In particular, OT enables the use of a lightweight in place of a full LLM forward pass, offering substantial computational savings. This substitution is especially valuable for enhancing translation robustness without relying on prompt-level guardrails: as long as the embedding dimensionality is sufficiently high, OT matching effectively separates natural-language semantics from lean code representations. This delineation helps stabilize translation quality even under constrained model conditions.

7 Conclusion and Future Work

Our approach demonstrates that neural dynamic OT can serve as a viable bridge between natural-language and formal proof embeddings. Our methods can be expanded to any formal language with sufficiently large aligned corpora. Experiments that compare the mappings between different levels of formalisms can be a useful tool to categorize these approaches, just like how Optimal transport has been used to compare similarities of human languages [22]. Further work can come through drastically different fields, such as comparing different translations of well-known literature, for example, comparing the translations of "The Odyssey" from different authors, to measure overall accuracy in translation and common-ground for both.

Future work can refine the transport mechanism by enforcing geometric constraints that keep transported embeddings closer to the pretrained Lean manifold, reducing the instability introduced by approximate soft tokens. Another direction is to extend OT beyond token-level alignment by incorporating larger proof structures, such as proof-term DAGs so we can capture higher levels of reasoning. Finally, scaling to larger datasets and additional proof systems would test the generality of our approach and reveal broader patterns in how formal language encodes mathematical structure.

References

[1] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[2] Prithwish Jana, Kaan Kale, Ahmet Ege Tanriverdi, Cruise Song, Sriram Vishwanath, and Vijay Ganesh. Proofbridge: Auto-formalization of natural language proofs in lean via joint embeddings, 2025.

[3] Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models, 2023.

[4] Alex Gu, Bartosz Piotrowski, Fabian Gloeckle, Kaiyu Yang, and Aram H. Markosyan. Proofoptimizer: Training language models to simplify proofs without human demonstrations. In *Submitted to The Fourteenth International Conference on Learning Representations*, 2025. under review.

[5] Alexander Korotin, Dmytro Selikhanovych, and Evgeny Burnaev. Neural optimal transport. 2021.

[6] Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A natural language annotated lean 4 dataset. In *The Thirteenth International Conference on Learning Representations*, 2025.

[7]

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

[9] Philippe Rigollet and Austin J. Stromme. On the sample complexity of entropic optimal transport, 2022.

[10] Jean-David Benamou and Yann Brenier. A computational fluid mechanics solution to the monge-kantorovich mass transfer problem. *Numerische Mathematik*, 84:375–393, 2000.

[11] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

[12] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023.

[13] Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport, 2024.

[14] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[15] RÃ©mi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, AurÃ©lie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, LÃ©o Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[16] Pedro C. Álvarez Esteban, E. del Barrio, J. A. Cuesta-Albertos, and C. Matrán. A fixed-point approach to barycenters in wasserstein space, 2016.

[17] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.

[18] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[19] Arthur Paulino. LeanREPL.

[20] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[21] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing nlp, 2021.

[22] David Alvarez-Melis and Tommi S. Jaakkola. Gromov-wasserstein alignment of word embedding spaces, 2018.

8 Appendix

Appendix A: Pseudocodes

Algorithm 1: Pseudocode for practical inference pipeline

```
Data: Natural language theorem-proof pair $(T_{\text{NL}}, P_{\text{NL}})$
Data: Pre-trained encoder $\mathcal{E}_{\theta_0}$ (AI-MO/Kimina-Prover-Distill-1.7B)
Data: Learned Neural OT vector field $v_\theta : \mathbb{R}^d \to \mathbb{R}^d$, $d = 2048$
Data: Language model decoder $\mathcal{D}_\phi$ with embedding dimension $d$
Data: Lean4 REPL verifier $\mathcal{V}$
Result: Verified Lean4 code $\text{code}_{\text{Lean}}$ and validation status valid
// Stage 1: Natural Language Embedding
$\mathbf{x}_{\text{NL}} \leftarrow \text{Tokenize}(T_{\text{NL}}, P_{\text{NL}})$;
$\mathbf{E}_{\text{NL}} \leftarrow \mathcal{E}_{\theta_0}(\mathbf{x}_{\text{NL}}) \in \mathbb{R}^{n \times d}$ ;                // $n$ tokens, dim 2048
;
// Stage 2: Embedding Translation via Neural OT
for $i = 1$ to $n$ do
    $\mathbf{e}_{\text{Lean}}^{(i)} \leftarrow \mathbf{e}_{\text{NL}}^{(i)} + v_\theta(\mathbf{e}_{\text{NL}}^{(i)})$ ;                // Transport to Lean space
end
$\mathbf{E}_{\text{Lean}} \leftarrow [\mathbf{e}_{\text{Lean}}^{(1)}, \ldots, \mathbf{e}_{\text{Lean}}^{(n)}] \in \mathbb{R}^{n \times d}$;
;
// Stage 3: Soft-Token Decoding
$\mathbf{p} \leftarrow$ Embed("Translate to Lean4:") ;                // Prompt embedding
$\mathbf{H} \leftarrow [\mathbf{p}; \mathbf{E}_{\text{Lean}}]$ ;                // Concatenate prompt and soft tokens
$\mathbf{y} \leftarrow \mathcal{D}_\phi(\mathbf{H})$ ;                // LLM forward pass
tokens $\leftarrow \text{ArgMax}(\mathbf{y})$ ;                // Decode to discrete tokens
;
// Stage 4: Post-Processing and Verification
$\text{code}_{\text{Lean}} \leftarrow$ ExtractCode(tokens);
valid $\leftarrow \mathcal{V}(\text{code}_{\text{Lean}})$ ;                // REPL verification
;
return $code_{Lean}$, $valid$
```

Appendix B: Autoformalization Attempts

25240th training example from Herald Dataset

Theorem: Range of Doubling Function Equals Set of Even Elements in Semirings : For any semiring α, the set of all elements of the form $2x$ for some $x \in \alpha$ is precisely the set of all even elements in α. In other words, the range of the function $x \mapsto 2x$ is equal to the set of all elements $a \in \alpha$ such that a is even.
Proof:[Redacted for brevity]
Our method:

```
import Mathlib
```

```
/- Range of the function x -> 2x equals the set of all even elements in a semiring -/
theorem range_doubling_eq_even_set (alpha : Type*) [Semiring alpha] :
  (range (fun x => 2 * x)) = {x | there exists r, x = 2 * r} := by
  ext y
  simp
  constructor
  · -- Assume y in range
    rintro ⟨x, hx⟩
    use x
    all_goals linarith
  · -- Assume y is even
    rintro ⟨r, hr⟩
    use r
    all_goals linarith
```

Baseline:

```
The error message "Lean process timed out’ has timed out’
indicates that Lean is timing out an error in the Lean process.
This error occurs because the code tries to run a Lean proof,
but the process has timed out ’has timed out’ but the error message
"Lean process timed out’ has timed out’ is stuck trying to prove the goal
‘False‘ in the second part of the proof.
The goal is ‘False‘ at the momentwhen the proof reaches
‘linarith‘in the second part of the proof. The error message
"Lean process timed out’has timed out’ indicates that the Lean process
is stuck trying to prove False. The error occurs at the line ‘linarith’
in the second part of the proof. The error message "Lean process timed out’has
timed out’ indicates that the Lean process has timed out’and has timed out’
indicates that the Lean process is stuck trying to prove the goal ‘False‘.
This error message appears when the proof finishes with
‘linarith‘ and the second part of the proof, which is trying to
prove the goal ‘False‘. This is happening at the line ‘linarith‘.

The original proof attempts to prove the theorem
‘range_of_f : (range_of_f : Function R → Set R) = Set.range (fun x => 2 *x) =
Set.range (fun x => 2*x)‘.
The theorem states that the rangeof the function \(f(x) =2x\)
is equaltothe setofall even numbers.

The proof uses the ‘ext y‘ tactic to splitthe goalintore range_of_f:
‘range_of_f =Set.range (fun x => 2*x)‘ into two implications.
1. The first direction: ‘range_of_f in Set.range (fun x => 2 *x)‘
(range_of_f is contained in the range of the function).
2. The second direction: ‘Set.range (fun x => 2*x) in range_of_f‘
```

```
(range_of_fis containedin the rangeof the function).

The error is in the second part ofthe proof. The second partofthe proofis:
‘intro y
intro hy
use y/2
all_goals
  simp only [Set.mem_setOf_eq]
use y/2
all_goals
  simp only [Set.mem_setOf_eq]
  use y/2
  all_goals
    simp only [Set.mem_setOf_eq]
    use y/2
    all_goals
      simp only [Set.mem_setOf_eq]
      use y/2
      <;> ring_nf
      <;> linarith
```

22758th training example from Herald Dataset

Theorem: Cardinality of Left Set Divides Cardinality of Binary Image if Partial Applications are Injective and Disjoint : For any types α, β, and γ with a decidable equality on γ, and for any binary function $f : \alpha \to \beta \to \gamma$, finite sets $s \subseteq \alpha$ and $t \subseteq \beta$, if each partial application $f(\cdot, b)$ is injective for all $b \in t$, and the images of s under these partial applications are pairwise disjoint, then the cardinality of s divides the cardinality of the binary image $f(s, t)$, i.e., $|s| \mid |f(s, t)|$.
Proof:[Redacted for brevity]
Our method:

```
import Mathlib

/- For any types \(\alpha\), \(\beta\), and \(\gamma\) with a
decidable equality on \(\gamma\), and for any binary function
\(f : \alpha \to \beta \to \gamma\), finite sets \(s \subseteq \alpha\)
and \(t \subseteq \beta\), if each partial
application \(f(\cdot, b)\) is injective for all \(b \in t\), and the
images of \(s\) under these partial applications are
pairwise disjoint, then the cardinality of \(s\) divides the cardinality
of the binary image \(f(s, t)\), i.e., \(|s| \mid
|f(s, t)|\). -/
theorem set_cardinality (f :  ×  → ) (s : Finset ) (t : Finset )
(h_inj :  b  t,  a a, f a b = f a b → a = a)
```

```
(h_disj : Disjoint (s × t) (s × t)) : (s.card)  (f ’’ (s × t)).card := by
  have h1 : Disjoint (s × t) (s × t) := h_disj
  have h2 : (f ’’ (s × t)).card = (s.card) * (t.card) := by
    have h3 : Disjoint (s × t) (s × t) := h1
    have h6 :  b  t,  a a, f a b = f a b → a = a := h_inj
    rw [Finset.card_image_of_injOn]
    · -- Show that f is injective on s × t
      intro b hb a a h
      have h8 : f (a.1) b = f (a.1) b := by simpa using h
      apply h6 b hb a a
      all_goals tauto
    · -- Show that the image is disjoint
      intro a ha a ha’ h
      simp at ha’ h
      tauto
  rw [h2]
  use t.card
  all_goals ring
```

Baseline:

```
ERROR [No Lean code found in output]
```

Appendix C: LLM Prompts

Autoformalization Prompt

```
"""
user_prompt = f"""Translate the following proof in Lean 4.
Here are some examples of similar problems and their solutions:

{examples_in_prompt}

Now, solve the following problem:
# Problem:
{informal_statement}

# Informal Proof:
{informal_proof}

Now autoformalize it in Lean 4 with a header.

<formal_proof>

</formal_proof>
```

```
Soft virtual tokens representing the target Lean 4 solution are provided
as contextual embeddings - use these to guide your formalization.
"""
```