Lean4Math: Using Neural Methods To Improve Math Reasoning in LLMs

Michael Souliman
Stanford University
Department of Computer Science
msoul@stanford.edu

Willy Chan
Stanford University
Department of Computer Science
willyc@stanford.edu

Jakob Nordhagen
Stanford University
Department of Computer Science
jakobn@stanford.edu

Abstract

Recently, large language models (LLMs) have recorded impressive performance and versatility on neural machine translation tasks. As such, LLMs show considerable potential for autoformalization, the task of translating proofs from informal mathematical language (IL) into formal mathematical language (FL). A success in this task would have significant and far-reaching implications, potentially allowing for the formalization of all known mathematical textbooks, as well as establishing an avenue for compute-driven advancement of the world's mathematical knowledge. However, vanilla LLMs currently lack robust quantitative reasoning capabilities, causing them to perform poorly on complex math tasks such as autoformalization and the downstream task of formal theorem proving. In this paper, we explore several approaches to fine-tuning LLMs to reason mathematically. Specifically, we test the extent to which backtranslation is effective as a way to mitigate the problem of lacking labeled data which has hindered advancement in autoformalization. We investigate several dataset generation strategies: on-the-fly (online) backtranslation, distilled (offline) backtranslation with few-shot amplification, considering proofs line-by-line, and incorporating proof state information. We find that our methods lead to nontrivial improvements in performance in autoformalization as measured on the ProofNet benchmark [2], showing promise for formal theorem proving and beyond.

1. Introduction

Neural machine translation (NMT) has been an area of active research since the early days of machine learning and neural network architecture development [7] [24].
Autoformalization can be interpreted as a special case of NMT, where the task is to translate a theorem (or any other mathematical statement) from informal language, denoted IL, to formal language, denoted FL. Here informal language is in the form of text and LaTeX, and formal language is specified as Lean code. Lean4 is the latest version of Lean, an open-source theorem prover and programming language designed to encode mathematics for the purpose of tasks such as automated theorem proving [6].

1.1. Reasoning & Autoformalization

The implications of fruitful research on autoformalization are significant and far-reaching. The emergence of an agent capable of accurate autoformalization would allow for automation of the laborious task of formalizing mathematics, massively decreasing the prohibitive cost of current efforts [13]. Such a development could revolutionize the field of automated theorem proving, making systems more accessible and effective on a broader range of mathematical problems. Further, this research contributes to the broader understanding of how LLMs can be fine-tuned for other specific, complex reasoning tasks, opening up new possibilities for their application in various scientific and mathematical domains such as program synthesis and formal verification. Finally, success in autoformalization would unite semantic understanding of natural language and grasp of formal reasoning, representing a remarkable achievement in LLM capability at large.

Despite considerable advancements in recent years, standard LLMs still struggle with complex quantitative reasoning and, by extension, with tasks like autoformalization and theorem proving. These limitations highlight a gap in the ability of current models to understand and translate complex mathematical concepts and proofs from natural language to formal logic. We seek progress in this area by explicitly fine-tuning models for autoformalization.

However, fine-tuning LLMs for the autoformalization task is considerably more difficult than for other translation tasks due to the lack of existing high-quality data. Mathematical programming languages such as Lean are not particularly prevalent, and availability of formal data, especially parallel corpora that contains matching (IL, FL) pairs, is a huge limiting factor. The sum total of formal language data in existence is a tiny fraction of the size of modern LLMs' training datasets, and there is essentially zero paired informal and formal data outside of hand-curated benchmarks such as ProofNet and MiniF2F [2] [26].

1.2. Backtranslation for Data Augmentation

Various techniques have been explored for data augmentation in cases where labeled training data for supervised learning is scarce. Backtranslation, a method for generating artificial training data, is one of the more prominent and proven methods for unsupervised NMT [16]. Given a monolingual dataset in the target language, backtranslation proceeds as follows:

1. Use some pretrained model to generate translations in the reverse direction, i.e. from the target language to the source language.

2. Assemble a parallel corpora made up of these (synthetic source, ground-truth target) pairs.

3. Fine-tune or train on the generated dataset, teaching a model to translate the synthetic source examples back to the target language.

Backtranslation is the underlying method we explored for generating our datasets, allowing us to fine-tune models in a semi-unsupervised fashion while only using formal-language data.

1.3. Relevance of Interactive Theorem Provers

Additionally, the use of Interactive Theorem Provers (ITPs) in extracting datasets of intermediate proof steps presents a novel opportunity for the task of autoformalization. By informalizing intermediate statements in a proof sequentially, we can generate a dataset that considers the individual steps undertaken by ITPs. This data can then be used to train an LLM, replicating in some sense the sequential reasoning performed by ITPs. This line-by-line informalization method is novel in the field of autoformalization.

1.4. Contribution

Our approach to enhancing LLMs for autoformalization and formal theorem proving represents a shift in the methodology of automated theorem proving. By focusing on individual proof lines and using backtranslation to generate various datasets that pair natural language proofs with formal statements, we aim to bridge the gap between natural language understanding and formal logic reasoning. This research not only advances the field of automated theorem proving but also contributes to the broader goal of creating models capable of mathematical reasoning. Such a development could expand the field of mathematics in a way that has never been seen before.

We introduce a new autoformalization dataset, **AI4M**, which pairs statements within natural language proofs with their equivalent in the Lean theorem proving language. This dataset serves two primary purposes: firstly, to train an autoformalizer on individual statement pairs, and secondly, to fine-tune an LLM on the state of the proof before and after a specific tactic is applied. This approach aims to closely mimic the reasoning process in theorem proving, enhancing the LLM's ability to understand and generate formal proofs.

To assess the effectiveness of our autoformalizer and theorem prover, we employ the MiniF2F and ProofNet [2] autoformalization benchmarks. These benchmarks are critical in evaluating our approach's ability to translate natural language theorems into formal language proofs accurately and efficiently. The success of our model on these benchmarks would demonstrate a significant advancement in the field of automated theorem proving.

In conclusion, our approach to enhancing LLMs for autoformalization and formal theorem proving represents a significant shift in the methodology of automated theorem proving. By focusing on individual proof lines and utilizing a dataset that pairs natural language proofs with formal statements, we aim to bridge the gap between natural language understanding and formal logic reasoning. This research not only advances the field of automated theorem proving but also contributes to the broader understanding of how LLMs can be tailored for specific, complex applications, laying groundwork for breakthroughs in various areas of scientific and mathematical research.

2. Related Works

The idea of solving theorem-proving problems—and, more generally, quantitative reasoning problems—with neural networks has been explored with interest during the last decade, fueled in no small part by the revolution of the transformer architecture [20] and the subsequent explosion of LLMs. While LLMs have performed well on various natural language understanding tasks as evaluated by benchmarks such as MMLU [10], they have continued to struggle with tasks that require deeper quantitative reasoning [15]. Moreover, while strides have been made in using deep learning for symbolic mathematics [14], propositional logic [8], and differential systems [5], the field of autoformalization and theorem proving has generally not seen proportional advances.

Autoformalization saw its first significant experiments in 2018 with the use of long short term memory networks (LSTMs) to generate statements in Mizar [22]. Most recently, Wu et al. laid important groundwork for autoformalization with LLMs, showing promising results using naive few-shot learning on the autoformalization task, in their case translating English to Isabelle code [23].

In the realm of NMT, numerous methods have been explored for mitigating the issue of limited training data. Backtranslation saw use for NMT as early as 2018 [19]. Later, Han et al. (2021) demonstrated the effectiveness of backtranslation for French-to-English translation, using the unsupervised method to new train state-of-the-art NMT models on top of transformer-based baselines [9]. Azerbayev et al. (2023) ran preliminary experiments using distilled backtranslation for autoformalization, using OpenAI's code-davinci-002 model to create synthetic training data [2].

The development of LeanDojo [25] has been crucial to efforts in formal theorem proving and has potential for autoformalization. LeanDojo is an open-source Lean playground and a benchmark of 98,734 Lean theorems that can act as a robust formal theorem proving dataset for fine-tuning LLMs. ReProver, a retrieval-augmented language model that outperforms both tidy and GPT-4, was also introduced along with LeanDojo [25]. While retrieval-augmented models like ReProver demonstrate the potential of using formal corpora to enhance language models, they still rely heavily on automated theorem provers, which limits its exposure to the underlying reasoning patterns and sequential steps that human mathematicians actually follow. Without explicitly modeling the intermediate reasoning process, the flexibility and explainability of integrated theorem proving systems is restricted. Additionally, overdependence on first-order logic hampers the expression of more complex proofs. To enable language models to develop richer reasoning abilities and generalize beyond their training proofs, our line-by-line approach looks to directly model the incremental deduction used by interactive theorem provers.

Researchers have traditionally focused on combining automated theorem prover and language models that struggle with quantitative reasoning. This focus has limited the types of theorems that can be expressed and solved, as well as the efficiency and applicability of these models in more complex, real-world scenarios. Our research takes a different approach. Instead of relying on automated theorem provers, we propose the integration of interactive theorem provers with LLMs by using the sequential nature of both ITPs and LLMs. By fine-tuning LLMs on intermediary statements extracted from the LeanDojo dataset, we aim to learn more about how formal theorem provers reason and abstract that reasoning into an LLM.

Lastly, the development of benchmarks for performance on reasoning tasks has accelerated development in the area of teaching LLMs to reason effectively [2] [26] [21] [10]. For a comprehensive overview of relevant benchmarks, see Section A.

3. Methods & Design

We set out to train an LLM on the formalization direction of the translation task, i.e. translating informal language (IL) to formal language (FL) in the form of Lean mathematical code. Given the very limited amount of labeled data or pairs of matching (IL, FL) examples to use as training data, we used backtranslation on custom-created datasets, using several different generation methods to create and evaluate distinct synthetic datasets. In general, we started with data in FL and translated in the reverse direction (FL $\rightarrow$ IL) to generate parallel corpora and produce our datasets. We refer to this process as informalization.

3.1. On-the-fly Backtranslation

The first training method we explored for the purpose of data augmentation was a form of backtranslation that we implemented within a custom training loop. This approach had no concept of separate teacher and student models; instead, we attempted to train the same model on both directions of the translation task at once, essentially having it generate its own training data during each step. Concretely, during each training step:

1. Translate a batch of FL examples to IL.
2. Translate the synthetic IL examples back to FL, using the (synthetic IL, FL) pairs as labeled training data.
3. Compute the loss of the generated FL translations compared to the original FL examples.
4. Backpropagate and update model weights.

The primary advantage of this method was its self-sufficient nature, allowing us to sidestep the problem of limited labeled data without incurring large API costs. However, the method suffered from the relatively small model used in our experiments (GPT-2 with 124M parameters). Intuitively, GPT-2's small size meant that it could not be very effective at generating synthetic informalizations. To our surprise, the on-the-fly backtranslation method showed a huge decrease in evaluation loss after only a few hundred training steps, considerably outperforming our expectations. However, we eventually encountered a plateau in performance gains due to the inherent limitations of the baseline model's capabilities. Due to a lack of the necessary computational resources, we were ultimately unable to validate this method with a larger model such as Llemma-7B [3] or Google's Gemma-7B. We hypothesize that using a larger and more capable pretrained baseline could have produced even more promising results.

3.2. Backtranslation with Few-shot Prompting

In light of this limitation, we shifted focus to distilled backtranslation, wherein we used a far more powerful pretrained model such as GPT-4 [17] as the "teacher" model, feeding it a dataset of FL examples and using it to create matching (synthetic) IL examples. Here we leveraged the fact that LLMs are excellent at in-context learning [4]. We used a few-shot prompt that prefixes the informalization prompt with six labeled examples of FL-to-IL translations. We then used this few-shot amplification method to create an entire labeled dataset for training instead of creating the data on-the-fly. This switch in focus led to final results that performed better given our small model size.

Specifically, we extracted theorems from the MathLib4 dataset [1], used a regular expression to parse each .lean file into individual theorems, and then used GPT-4 with few-shot prompting to informalize each individual theorem. The prompts we used can be found in Section C of the appendix.

Using our regular expression, we parsed over 100,000 theorems which we could later informalize and use as training data for fine-tuning. However, due to the relatively high cost of using OpenAI's GPT-4 API, we were only able to informalize a small subset of those proofs. This method worked well for our experiments because we could use the API and not have to rely on the compute resources we had locally, effectively mitigating the main problem with on-the-fly backtranslation (model size).

Within the scope of distilled backtranslation, we explored and compared several ways of looking at proofs.

3.2.1 Entire Proofs

Our first method follows the methods used in ProofNet [2], where we informalize on the entire theorem while including the entire proof as context in the prompt, generating just the theorem statement in natural language. For this method, we used a 6-shot prompt to generate a dataset of 348 training pairs.

3.2.2 Individual Lines

Our second method aims to informalize just the theorem statement along with each individual tactic within a proof. By informalizing each individual tactic, we wanted to test if each tactic was able to better explain what was happening in each tactic and therefore better explain the proof as a whole. This method was focused on improving the translations between the entire Lean theorem and the entire natural language proof, not just the theorem statements.

For these informalizations, we used the LeanDojo dataset [25], which came already parsed into the different tactics taken. In the prompt, we included not only the formal theorem statement but also the state of the proof before the tactic was applied and the state after the tactic was applied. We used a zero-shot prompt with these tuples of (stateBefore, tactic, stateAfter) to generate a dataset of individually translated tactics.

3.3. Gathering Data from Regular Expressions

We also prepared a large dataset leveraging only regular expressions for capturing specific lines/keywords in tactic scripts. Utilizing regular expressions for parsing LeanDojo [25] proof tactics allowed us precise control over the selection of specific tactic lines and proof methodologies, enabling targeted improvements in specific areas of interest. For example, by looking for specific keywords such as "induction" or "def" within tactic scripts, we can infer that the proof is beginning a proof on induction or introducing a definition respectively. An example of one of the scripts we used to parse data from LeanDojo can be found in Section B.

3.4. Hyperparameter Tuning

Our optimization strategy concentrated on two critical hyperparameters: batch size and learning rate. These parameters were also needed in fine-tuning and maximizing the model's performance for the task of generating informalization statements from formal proofs.

To identify the optimal settings for these hyperparameters, we systematically varied the batch size and learning rate, analyzing the impact of each configuration on the model's evaluation loss. This iterative process was designed to hone in on the configurations that minimized eval loss, indicating a more effective model performance.

The effectiveness of each hyperparameter setting was assessed by observing changes in evaluation loss. This direct measure provided an indication of how well the model was learning and generalizing from the training data to the validation set. Specifically, the exploration of batch size and learning rate settings led to refined choices that balanced training efficiency with model accuracy. References to the results of these adjustments, illustrating the eval loss across varying hyperparameter settings, are provided in Figures 2 and 3 respectively.

3.5. AI4Math Dataset

The integration of data collected through these methodologies culminated in the formation of a comprehensive dataset. This dataset was pivotal in training our model, ensuring a broad spectrum of proof tactics and informal statements was represented. We call this resulting collection of datasets the "AI4Math" Dataset. The total number of tokens from each collection method is provided in Table 1. The following datasets are labeled according to the composition of their contents:

1. "MMA Train" is a large, multilingual, and multi-domain baseline dataset [11] which we aim to compare against.

2. "GPT-4 MathLib4 (Full Proof)" utilizes regular expressions to parse the MathLib4 dataset into individual theorems and employ GPT-4 with few-shot prompting to informalize a selected subset. Inspired by ProofNet, we then utilized a 6-shot prompting technique to transform entire theorem statements and their proofs into natural language. The full procedure is outlined in 3.2.1.

3. "GPT-4 LeanDojo (Individual Tactics)" contains not only informalization of theorem statements but also each discrete tactic within the proofs from the LeanDojo dataset, which includes data on the proof's state before and after each tactic's application. Employing a 0-shot prompting approach, we generated a dataset that contains the informal translations of each tactic, paired with their corresponding before and after states. The full procedure is outlined in 3.2.2.

4. "Regex-Parsed LeanDojo Proofs" uses regular expressions to parse specific tactics from the LeanDojo proofs for targeted model refinement. The procedure is described in section 3.3.

5. "On-the-Fly Backtranslation" is comprised of formal-informal pairs created as a result of the backtranslation procedure described in Section 3.1.

Data Collection Method	Token Count
MMA Train	10,916,097
GPT-4 MathLib4 (Full Proof)	71,550
GPT-4 LeanDojo (Individual Tactics)	1,754
Regex-Parsed LeanDojo Proofs	124,782

Table 1. Token distribution across dataset generation methods. For each dataset generation method, we compile formal and corresponding generated informal statement data into pairings. The model is trained and evaluated on each individual dataset to assess method-specific performance, with the key metric being evaluation loss on ProofNet's test dataset. The table presents the number of tokens contributed by each method to the overall dataset, providing insights into the diversity and scale of training data available for model optimization.

4. Evaluation

We primarily quantify the results of our methods by evaluating our fine-tuned models for accuracy on ProofNet's test dataset, with the goal of measuring the performance gain over their initial pretrained states. By reserving a subset of data exclusively for testing, we ensure that our models undergo a fair evaluation. In this way, we are able to test different dataset generation methods against each other in a precise manner.

For our experiments, we minimized the evaluation loss and used the ProofNet test dataset as our evaluation set. All of our models used GPT-2 as the base model and fine-tuned for 3 epochs. We included the results of fine-tuning on the training set of MMA, a dataset that also uses GPT-4 to create the formal-informal pairs that match the approach we took with the full-proof informalizations. [11] Table 2 notes the evaluation loss measured.

We have two major observations from these performance differences.

4.1. Backtranslation

When generating the distilled backtranslation dataset for the individual tactics, a review of the data showed that the model trained on our On-the-Fly Backtranslation dataset considerably outperformed the GPT-2 baseline, but its performance plateaued below that of the MMA dataset. We hypothesize that this is due to model size, as both of our best performing methodologies used GPT-4 to generate data for backtranslation.

Specifically, one method we attempted with prompt engineering was to have GPT-4 informalize the proof as a whole, then pair the individual tactics to a line in the translated natural language proof. However, this method was much more costly, at $0.15 per proof informalized and split, compared to the $0.05 that it cost to just informalize the whole proof. Given that the model performed much better than the baseline with far fewer tokens, we hypothesize that this is method has significant potential and should be explored in the future with proper funding.

Figure 1 illustrates the general flow of data in our main backtranslation method for fine-tuning as well as our evaluation pipeline.

4.2. Parsing Lean via Neural Methods

Another observation was that GPT-4 LeanDojo (Individual Tactics) performed significantly worse compared to GPT-4 MathLib4 (Full Proof). The latter dataset also outperformed the model trained on MMA, despite MMA being over 150 times larger. This may be a result of our informalization prompt being much longer and using a 6-shot approach; this means our informalizations cost five times more than those generated for MMA, which had roughly 332,000 theorems informalized for $3,500 total [11].

4.3. Individual Tactics

We saw similar performance between Regex-Parsed LeanDojo Proofs and GPT-4 LeanDojo (Individual Tactics).

Both of these datasets consider the Lean tactic scripts line by line, which suggests that the neural models we trained generally struggle to translate the individual tactics in formal proofs without the context of the proof as a whole.

Fine-Tuning Dataset	Eval Loss
GPT-2 Baseline	75.1839
GPT-4 LeanDojo (Individual Tactics)	5.1287
Regex-Parsed LeanDojo Proofs	4.4709
On-the-Fly Backtranslation	4.032
MMA Train	3.8791
GPT-4 MathLib4 (Full Proof)	3.1209

Table 2. Evaluation loss after fine-tuning GPT-2 on each dataset for 3 epochs. The evaluation dataset was the ProofNet Test Dataset. The right column is the evaluation loss of the fine-tuned model when evaluated on the ProofNet Test Dataset.

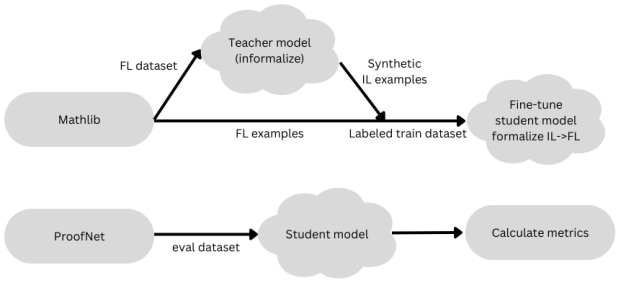

Figure 1. Diagram of the data flow in our main training method and in our evaluation pipeline. Leveraging the MathLib and ProofNet datasets, a "Teacher model" translates formal theorems into synthetic informal language (IL) examples, which are then backtranslated to augment a training dataset with new formal language (FL) examples. A "Student model," more compact than the teacher, is fine-tuned on this enriched dataset to improve its translation from IL to FL. Subsequently, the student model's proficiency in autoformalizing informal language to formal expressions is rigorously assessed using ProofNet's evaluation dataset, with performance metrics providing insight into the model's accuracy and ability to generalize.

5. Discussion

5.1. Limitations

While we found positive results for our datasets, we noted a few limitations with our method.

5.1.1 Compute & Cost

One of the primary limitations we encountered in our study was the constraint imposed by the size of the model we could feasibly train. Specifically, our "On-the-Fly" backtranslation process was hindered due to model size, as we could not employ larger models which might have provided more nuanced and meaningful translations.

Further, our fine-tuning was exclusively conducted using GPT-2. In future research iterations, we anticipate that employing more powerful models—such as Llama 7B, Mistral 7B, or Llemma 7B for fine-tuning could lead to more favorable outcomes. This enhancement could not only improve the quality of informalizations but might also alter our dataset's comparative performance against benchmarks like the MMA dataset.

Cost was another factor constraining our dataset's expansion. Informalization costs escalated when employing more sophisticated prompt engineering methods with GPT-4, particularly when translating proofs in their entirety and then segmenting them into individual tactics. Although this approach was notably more expensive—$0.15 per proof versus $0.05 for a complete proof informalization—its potential was evident as it surpassed our baseline performance with significantly fewer tokens. This indicates that despite the higher costs, the quality of the data generated through this method was superior. Given this method's demonstrated potential, we believe that it warrants further exploration, provided that adequate funding is available to support the increased costs. The allocation of such resources could be pivotal in scaling the dataset and, by extension, improving the efficacy of the autoformalization process.

5.1.2 Interactive Theorem Proving

In our current research, we concentrated on autoformalizing theorem statements rather than generating complete formal proofs. This focus was primarily driven by our goal to streamline the process of translating natural language into formal language theorems, which are integral to the groundwork of interactive theorem proving (ITP). However, a significant next step in our exploration involves assessing whether the entire proofs or tactic scripts, when generated by our model, can be successfully compiled within the Lean4 environment.

The ability for a model to generate not just the theorem statements but also the accompanying proofs would mark a substantial leap forward in automating the theorem proving process. It would not only enhance the efficiency and accessibility of formal verification but also potentially transform how complex mathematical and logical reasoning is conducted in computational settings. Thus, the next phase of our research will be dedicated to testing the compatibility and correctness of model-generated proofs with Lean4,

aiming to bridge the gap between automated theorem generation and its practical application in ITP systems. This endeavor will necessitate a deeper integration of our model's outputs with the stringent syntax and logical framework of Lean4, posing a challenging yet essential milestone towards fully automated theorem proving.

5.2. Future Implications

The success of this research could have a profound impact across multiple domains. Training an agent to perform autoformalization accurately would greatly diminish the time and resources currently needed to convert vast amounts of natural-language mathematical proofs into formal language. Such an advancement would not only make automated theorem proving more accessible and efficient but could also accelerate related areas such as formal verification and program synthesis. Looking ahead, a proficient autoformalizer has the potential to render all mathematical knowledge, presently recorded primarily in natural language, programmable. This would substantially enhance the usability of interactive theorem proving systems for a broader range of users, thereby accelerating the expansion of human mathematical understanding.

6. Conclusion

In this study, we have introduced a novel approach towards enhancing large language models for the task of autoformalization, laying the groundwork for significant advancements in automated theorem proving. By focusing on fine-tuning LLMs to convert natural language proofs into their formal counterparts line-by-line, and by creating the AI4M dataset, we have not only provided a method that promises a deeper understanding of mathematical proofs but also demonstrated the potential of this method through substantial improvements in model performance on benchmarks like ProofNet.

Looking ahead, the implications of our work extend far beyond the immediate results. By improving the autoformalization capabilities of LLMs, we set the stage for a future where the large existing library of natural language mathematics can be made accessible via a formal translation. This could revolutionize fields that rely on precise mathematical reasoning, from formal verification to program synthesis, and make automated theorem proving a more integral part of the mathematical research process. As we continue to refine our methods and expand our datasets, we also have potential to further research that could one day enable LLMs to not just understand but also to contribute to advancing the frontier of human knowledge in mathematics and beyond.

References

[1] The lean mathematical library. *CoRR*, abs/1910.09336, 2019. 4

[2] Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 2023. 1, 2, 3, 4, 8

[3] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics, 2023. 3

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. 4

[5] Francois Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. In *International Conference on Learning Representations*, 2021. 2

[6] Leonardo Mendonça de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *CADE*, 2021. 1

[7] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. pages 1723–1732, 01 2015. 1

[8] Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. Teaching temporal logics to neural networks. In *International Conference on Learning Representations*, 2021. 2

[9] Jesse Michael Han, Igor Babuschkin, Harrison Edwards, Arvind Neelakantan, Tao Xu, Stanislas Polu, Alex Ray, Pranav Shyam, Aditya Ramesh, Alec Radford, and Ilya Sutskever. Unsupervised neural machine translation with generative language models only, 2021. 3

[10] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *CoRR*, abs/2009.03300, 2020. 2, 3, 8

[11] Albert Q. Jiang, Wenda Li, and Mateja Jamnik. Multilingual mathematical autoformalization, 2023. 5

[12] Albert Q. Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu, and Mateja Jamnik. Thor: Wielding hammers to integrate language models and automated theorem provers, 2022. 8

[13] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *ACM Symposium on Operating Systems Principles*, pages 207–220, Big Sky, MT, USA, Oct. 2009. ACM. 1

[14] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020. 2

[15] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022. 2

[16] Xuebo Liu, Longyue Wang, Derek F. Wong, Liang Ding, Lidia S. Chao, Shuming Shi, and Zhaopeng Tu. On the complementarity between pre-training and back-translation for neural machine translation, 2021. 2

[17] OpenAI. Gpt-4 technical report, 2023. 4

[18] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020. 8

[19] Alberto Poncelas, Dimitar Shterionov, Andy Way, Gideon Maillette de Buy Wenniger, and Peyman Passban. Investigating backtranslation in neural machine translation, 2018. 3

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. 2

[21] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019. 3, 8

[22] Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. First experiments with neural translation of informal to formal mathematics. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics*, pages 255–270, Cham, 2018. Springer International Publishing. 3, 8

[23] Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models, 2022. 3, 8

[24] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016. 1

[25] 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. 3, 4

[26] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021. 2, 3, 8

A. Further Related Works

While general benchmarks for natural language understanding, such as the Massive Multitask Language Understanding (MMLU) [10] and General Language Understanding Evaluation (GLUE) [21] benchmarks, are powerful in understanding the reasoning capabilities of a model, they do not offer rigorous testing for theorem proving. The addition of the MiniF2F [26] and ProofNet [2] benchmarks have allowed for more rigorous testing by testing on formal Olympiad-level mathematics problems. These benchmarks are also crucial to the development of informal theorem proving models and the translation of both statements and proofs from natural language to formal math languages. The neural machine translation of informal to formal mathematics, termed Autoformalization, was first experimented with in 2018 with the use of LSTMs to generate statements in Mizar [22]. Since then, the use of LLMs for autoformalization has led to the development of the current state-of-the-art translation from informal math to Isabelle/HOL, a formal math language, on the MiniF2F benchmark [23].

The first exploration of using generative language models for this task of theorem proving is GPT-f, an automated prover and proof assistant that showed how transformers can be used for formal reasoning tasks [18]. One prominent theorem prover that built off of GPT-f is Thor, a system aimed at integrating language models with automated theorem provers [12], which was the state-of-the-art model for the MiniF2F benchmark [26] before the introduction of autoformalization. Automated theorem provers are based on first-order logic which limits the types of theorems they can express because these are automated procedures. However, Interactive Theorem Provers (ITPs) use higher-order logic that allows them to model theorem proving as a sequential decision process, similar to that of language models that sequentially output the most likely next token.

B. Regex-Parsed LeanDojo Proofs

We used the following regular expression to parse out specific patterns from Lean code:

```
patterns = {
    "induction": r"induction .+ with .+",
    "apply": r"apply .+",
    "rewrite": r"rw .+",
    "reflexivity": r"refl",
    "cases": r"cases .+",
    "introduce": r"intro .+|intros .+",
    "simplification": r"simp .+",
    "contradiction": r"contradiction",
    "exact": r"exact .+",
    "definition": r"def .+ := .+"
}
```

This allowed us to then format the following strings and informalize the proof:

```
"induction": "We're beginning
a proof by induction on {variable}.",
"apply": "Here, we apply the
theorem/lemma {theorem_name}.",

"rewrite": "We're rewriting part
of the expression using
{equality_statement}.",

"reflexivity": "This step concludes
that both sides of our equation
are identical.",

"cases": "We're breaking down
the problem into cases based on
{variable_or_condition}.",

"introduce": "We introduce new
variables {variable_names}.",

"simplification": "We simplify
the current expression or goal using
the simp tactic.",

"contradiction": "This step shows that
our assumptions lead to a contradiction.",

"exact": "Here, we provide the exact term
{term_name} that solves our current goal.",

"definition": "We define a function
{function_name} that takes {parameters}."
```

C. GPT4 Informalization Prompts

We used the following prompt for informalizing the MathLib4 theorems we extracted:

```
At the end of this explanation,
I will give you 2 things.
The first is a list of tuples
that are the translations of
entire proofs written in Lean,
which we will denote the formal
language, to plain English,
also known as natural language,
as tuples or pairs. This is not
an exhaustive list, these are
just examples of informalizations.
I will then have a proof written
in Lean represented as a string
following the newline character
after the list of pairs.
Give me the tuple pair of the
proof I give you written in Lean
and what you think their natural
language equivalent is given your
knowledge of Lean, formatted
using LaTeX. Do not output
anything else, just the python
tuple I requested. In your output
match the exact format "('formal',
'informal')" \n
[("theorem exists_le_sylow
{p : } {G : Type*} [group G]
{P : subgroup G} \n (hP :
is_p_group
p P) : \n  (Q : sylow p G), P  Q :=",
"Let $P$ be a $p$-subgroup of $G$.
Then $P$ is contained in a
Sylow $p$-subgroup of $G$."),
("theorem exists_eq_const_of_bounded
{E : Type u} [normed_group E]
\n [normed_space  E] {F : Type v}
[normed_group F] [normed_space  F]
\n {f : E → F} (hf : differentiable
 f) (hb : metric.bounded (set.range
f)) : \n  (c : F), f = function.const
E c :=", "Let $E$ and $F$ be
complex normed spaces and let
$f:E\to F$. If $f$ is differentiable
and bounded, then $f$ is constant.")
, ("theorem subset_of_open_subset_is_open
(X : Type*) [topological_space X]
\n (A : set X) (hA :  x  A,  U
: set X, is_open U  x  U  U
 A): \n is_open A :=","Let
$X$ be a topological space; let
$A$ be a subset of $X$. Suppose
that for each $x\in A$ there
is an open set $U$ containing
$x$ such that $U\subset A$.
Then $A$ is open in $X$."),
("theorem
is_multiplicative.eq_iff_eq_on_prime_powers
{R : Type*} \n [comm_monoid_with_zero
R] (f : nat.arithmetic_function R) \n
(hf : f.is_multiplicative) (g :
nat.arithmetic_function R) \n
(hg : g.is_multiplicative) : \n
f = g   (p i : ), nat.prime p
→ f (p ^ i) = g (p ^ i) :=",
"Two multiplicative functions $f,
g:\mathbb{N}\to R$ are equal if
```

```
and only if $f(p^i)=f(g^i)$ for
all primes $p$."), ("theorem
abs_sum_leq_sum_abs (n : ) (f :
 → ) : \nabs ( i in finset.range
n, f i)   i in finset.range n, abs
(f i) :=","If $z_1, \\dots, z_n$
are complex, then $|z_1 + z_2 +
\\dots + z_n|\\leq |z_1| + |z_2|
+ \\dots + |z_n|$."), ("theorem
distinct_powers_of_infinite_order_element
(G : Type*) [group G] (x : G) \n
(hx_inf :  n : , x ^ n  1) : \n
 m n : , m  n → x ^ m  x ^ n :=",
"If $x$ is an element of infinite order
in $G$, prove that the elements $x^n$,
$n\\in\\mathbb{Z}$ are all distinct.")] \n
{current_theorem_to_informalize}
```

D. Hyperparameter Tuning Figures

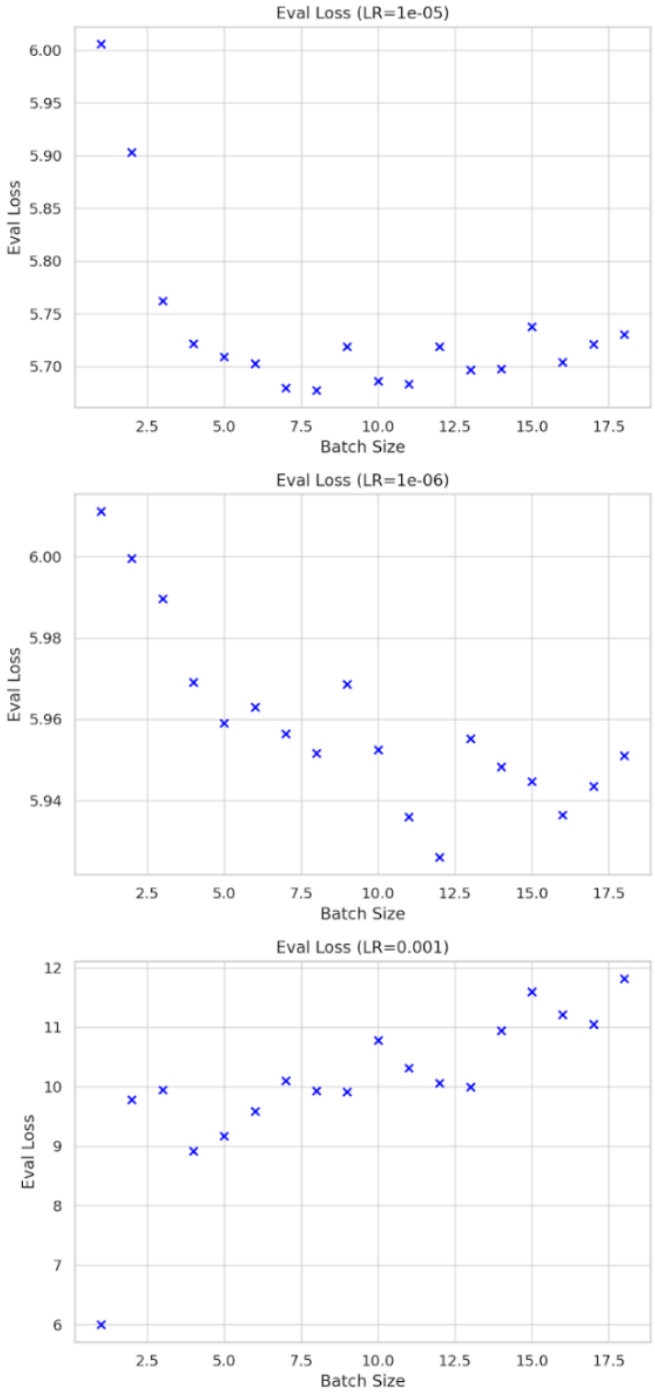

Figure 2. Evaluation Loss as a Function of Batch Size on Few-Shot GPT Informalization Dataset. This graph demonstrates how varying batch sizes influence the model's evaluation loss, indicating optimal batch size for minimized loss across varying learning rates.

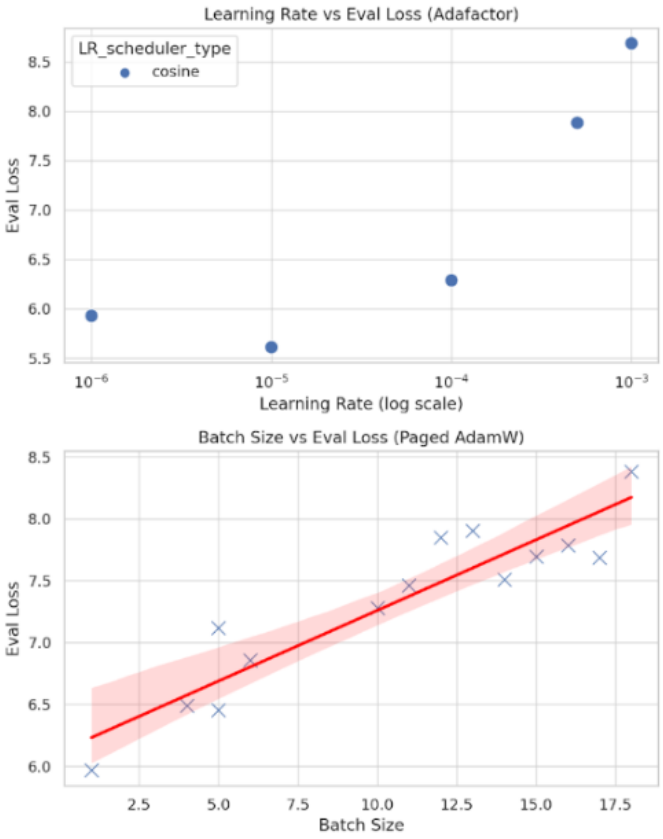

Figure 3. Loss Across Hyperparameter Optimization on GPT-4 dataset. Illustrates the results of varying batch size and learning rate across two different optimization methods: Adafactor and Paged AdamW.