Magnushammer: A Transformer-Based Approach to Premise Selection

Maciej Mikuła[1 * †] **Szymon Antoniak**[1 2 †] **Szymon Tworkowski**[1 3 †]

Bartosz Piotrowski[2] **Albert Qiaochu Jiang**[4] **Jin Peng Zhou**[3] **Christian Szegedy**[3]

Łukasz Kuciński[2 5] **Piotr Miłoś**[2 5] **Yuhuai Wu**[3]

[1]University of Warsaw [2]IDEAS NCBR [3]Google Research
[4]University of Cambridge [5]Polish Academy of Sciences

Abstract

We present Magnushammer: a novel approach to premise selection – a crucial task in automated theorem proving. Traditionally, symbolic methods that rely on domain knowledge and engineering effort are applied to this task. In contrast, this work demonstrates that contrastive training with the transformer architecture can achieve higher-quality retrieval of relevant premises, without the domain knowledge or feature engineering overhead. Magnushammer outperforms the most advanced and widely used automation tool in interactive theorem proving: Sledgehammer. On the PISA and miniF2F benchmarks Magnushammer achieves 59.5% (against 38.3%) and 34.0% (against 20.9%) success rates, respectively. By combining Magnushammer with a language-model-based theorem prover, we further improve the state-of-the-art proof success rate from 57.0% to 71.0% on the PISA benchmark. Moreover, we develop and open source a novel, large dataset for premise selection.

1 Introduction and background

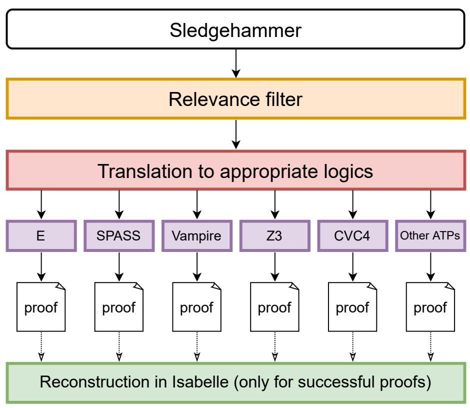
Modern mathematics development is gradual: it feeds upon a huge body of already established knowledge and constantly adds to it. Proving a mathematical statement requires retrieval of facts from the knowledge base that can advance the proof. In automated reasoning literature, this problem is known as *premise selection*, and many tools have been developed to tackle it [1, 23, 21, 3].

Proof assistants (aka *interactive theorem provers*, or ITPs) such as Isabelle [31], Lean [10], or Coq [4], are software tools designed to assist the development of formal proofs. They provide expressive language for the formalization of mathematical statements and proofs while verifying them formally.

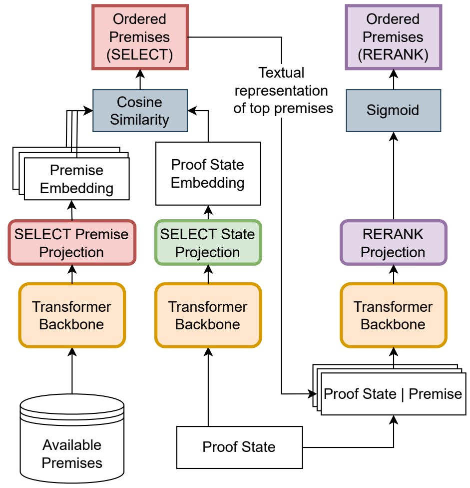
In Isabelle, theorems are proved sequentially: an initial *proof state* is obtained after the theorem is stated, and the proof state changes when the user provides a valid *proof step* (see Appendix A.1 for an example). Proof states contain information about the already established facts and the remaining goals to prove. Proof steps consist of *tactics*, which are optionally parametrized by *premises*. Tactics are theorem-proving procedures and can complete some proofs in one step provided with relevant premises. However, finding these premises is difficult: one needs to select a handful of relevant facts from the current proof context, which typically contains tens of thousands of them.

[*]Now at Google DeepMind.
[†]Equal contribution.

37th Conference on Neural Information Processing Systems (NeurIPS 2023) Workshop on MATH-AI.



(a) A call to Sledgehammer triggers the following sequence of steps: First, available facts are filtered based on their similarity to the conjecture. Then, the conjecture together with the selected facts (usually a few hundred in number) are translated to simpler logic used by the external provers (E, SPASS, etc.). Then, such problems are fed into each ATP separately. Finally, the premises used in the successful ATP proofs are used to reconstruct a proof inside Isabelle using its native methods.



(b) Given a proof state, we first retrieve the most relevant premises according to the cosine similarity of their embeddings with the proof state embedding (SELECT). We then re-rank these with a model that encodes each proof state and premise pair, outputting a relevance score (RERANK). The bulk of the architecture is a shared transformer model, in orange.

Figure 1: Overview of Sledgehammer (a) and Magnushammer (b).

Sledgehammer [32, 5] is a powerful automated reasoning tool for Isabelle. It belongs to a broader class of tools known as *hammers*, which integrate automated theorem provers (ATPs) into proof assistants to automate the process of constructing proofs. Sledgehammer has become an indispensable tool for Isabelle practitioners [32]. It allows for closing low-level gaps between subsequent high-level steps of proof without the need to memorize entire lemma libraries.

Sledgehammer is designed to first pre-select a number of relevant facts heuristically, translate them together with a conjecture to simpler logic, and try to prove the conjecture using strong, external ATPs (like E [40] or Vampire [22]). If successful, these provers generate complete proofs. They are, however, not trusted by Isabelle. Instead, the facts used in them are extracted and used to produce a proof *inside* Isabelle using its native methods. Up to this last step, known as *proof reconstruction*, Sledgehammer is essentially used as a premise selection tool. See Figure 1a depicting this process.

In this study, we provide a novel, generic, data-driven, transformer-based [45] premise selection tool: Magnushammer. In Section 2, we describe its architecture; in Section 3 we characterize the dataset extracted from Isabelle libraries for training it; finally, in Section 4, we demonstrate that Magnushammer achieves substantially better proving performance compared to Sledgehammer.

2 Magnushammer

The core idea of Magnushammer is to carry out premise retrieval in two stages: SELECT and RERANK. In the SELECT stage, it performs fast retrieval of the most relevant premises located in the neighbourhood of the current proof state in the common embedding space. In the RERANK stage, the retrieved premises are re-ranked with a more precise (but slower) scoring method that has access to the tokens of the proof state and premises. This hierarchical approach, which closely follows [29] and [16], is scalable to large formal libraries containing hundreds of thousands of facts. The Magnushammer's architecture is depicted in Figure 1b and outlined in Algorithm 2 (in Appendix C).

SELECT leverages *representation similarity* and is based on batch-contrastive training [2, 3, 14, 37]. It embeds premises and proof states into a common latent space and uses cosine similarity to determine their relevance. During inference, it requires only one pass of a neural network to compute the proof state embedding and dot product with cached premise embeddings. SELECT is hence fast and scalable to large sets of premises. In our experiments, there are between 30K and 50K premises in a typical proof state context, from which we select $K_S = 1024$ most relevant ones.

RERANK scores the relevance of the K_S selected premises for the current proof state by analyzing the $(\texttt{proof\_state}, \texttt{premise})$ pairs. RERANK is trained to output the probability of the `premise` being relevant to the `proof_state`. The K_S premises retrieved by SELECT are re-ranked with respect to these probabilities, and the final list comprises of the top K_R premises (we set $K_R = K_S$). Having both the premise and the proof state in a single input allows RERANK to be more accurate. However, at the same time, it is much slower, as each pair must be scored individually.

Training Magnushammer shares a transformer backbone with specialized linear projections on top (see Figure 1b). The backbone is pre-trained with a language modeling task on the GitHub and arXiv subsets of the Pile dataset [11]. Then, we train Magnushammer alternating between SELECT's and RERANK's objectives, using data consisting of $(\texttt{proof\_state}, \texttt{premise})$ pairs extracted with a procedure described in Section 3. Appendix B provides complete details of the training procedure.

SELECT is trained contrastively with a modified InfoNCE loss [44] using batches consisting of N proof states, N positive premises (one for each proof state), and additional M negative premises sampled from available facts that are not ground truth premises for any of the selected proof states. (This gives $N - 1 + M$ negatives per proof state in one batch; we typically use $M = 3N$.)

RERANK is trained using a standard binary classification objective. For each positive $(\texttt{proof\_state}, \texttt{premise})$ pair in the dataset, we construct 15 negatives from the most likely false positives returned by SELECT. Specifically, all the premises $\mathcal{M}$ that are facts that were never used as a premise for `proof_state`, are first chosen. Then, the top 1024 of $\mathcal{M}$ according to SELECT are selected, and 15 are sampled from them to construct negative training pairs.

Evaluation in Isabelle Given a proof state, a list of the k most relevant premises P is retrieved. We construct proof steps consisting of a tactic t and a subset of premises $S \subseteq P$. Such proof steps are executed in parallel, with a timeout of 2 seconds. The evaluation is successful if any of these proof steps completes the proof. For S, we pick the top i of P, where i's are consecutive powers of 2 up to 2^{10}, or 0 for tactics that do not accept premises. More details, including the set of tactics used, are presented in Appendix C. An example of a proof with tactics and premises is given in Appendix A.3.

3 Datasets

We created and released[3] a comprehensive dataset of textual representations for Isabelle's proof states and premises.To the best of our knowledge, this is the first high-quality dataset of this kind for Isabelle, and also the largest premise selection dataset overall. We used the two largest collections of Isabelle theories to create the dataset: the Archive of Formal Proofs and the Isabelle Standard library.

For every proof step in every proof from these collections, we extracted the preceding proof state and the premises used in the proof step; this was turned into $(\texttt{proof\_state}, \texttt{premise})$ pairs constituting training data points. We call this the HUMAN PROOFS LIBRARY (HPL) dataset. In addition, we used Sledgehammer to generate proofs that are different from the human ones by using potentially alternative premises. We refer to this as the SH partition, and its union with HPL is the MACHINE-AUGMENTED PROOFS LIBRARY (MAPL) dataset. Our datasets have 2 distinguishing features:

1. The human-originating dataset is augmented by alternatives generated with Sledgehammer, which results in a significantly larger and more diverse dataset. This also decreases the probability of sampling *false negatives* while training contrastively: a negative example $(\texttt{proof\_state}, \texttt{premise})$ may in fact be positive, but we just have not seen an alternative proof using `premise`. Generating multiple alternative proofs partially remedies this problem.
2. Both `proof_states` and `premises` are represented as "high-level" Isabelle's text instead of "low-level" logical formalism (like in [1]). This makes the data more suitable for language models, avoids feature engineering, and facilitates cross-proof-assistant pre-training [8].

[3]The data is available on HuggingFace:
`https://huggingface.co/datasets/Simontwice/premise_selection_in_isabelle`.

Table 1: Proof rates on the PISA benchmark. On the single-step task, Magnushammer outperforms both Sledgehammer and BM25 by a wide margin. On the multi-step task, Magnushammer combined with Thor achieves the state-of-the-art proof rate of 71.0%.

Setting	Method	Proof rate (%)
single	TF-IDF	31.8
	BM25	30.6
	OpenAI embed.	36.1
	Sledgehammer	38.3
	Magnushammer	**59.5**
multi	LISA	33.2
	Thor	57.0
	Thor + Magnushammer	**71.0**

Table 2: Proof rates on the miniF2F benchmark. On the single-step task, Magnushammer outperforms both Sledgehammer and a variant with additional heuristics [19]. On the multi-step task, Thor + Magnushammer obtains competitive results, significantly outperforming Thor + Sledgehammer.

	Method	Valid (%)	Test (%)
single	Sledgehammer	9.9	10.4
	Sledgehammer + heuristics	18.0	20.9
	Magnushammer	**33.6**	**34.0**
multi	Thor + Sledgehammer	28.3	29.9
	Thor + Sledgehammer + auto	37.3	35.2
	Thor + Magnushammer	36.9	37.3
	DSP [19]	**43.9**	**39.3**

4 Experiments

We evaluate Magnushammer on two established theorem-proving benchmarks using two different settings specified below: *single-* and *multi-step* settings. Our main result is that Magnushammer outperforms Sledgehammer by a large margin and, combined with Thor [18], sets a new state of the art on the PISA benchmark (71.0% from 57.0%). Evaluation details and ablations can be found in Appendices C and D, respectively. In particular, Figure C.5 shows how Magnushammer outperforms Sledgehammer across a broad spectrum of computational budgets.

Benchmarks and evaluation metrics For evaluation, we use PISA [17] and miniF2F [51] benchmarks. PISA contains problems randomly selected from the Archive of Formal Proofs; we use the same 1000 problems as Jiang et al. [18] for our evaluations. miniF2F consists of 488 high-school competition-level problems, split into validation and test set, each with 244 problems.

To evaluate the performance, we measure *proof success rate*: the percentage of successful proofs. A proof is successful if it is formally verified by Isabelle.

Single-step setting In this setting, we check if a theorem can be proven in a *single step* by applying premises retrieved by the evaluated premise selection method. To this end, we generate $|\mathcal{T}| \times |K|$ proof steps by combining each tactic $t \in \mathcal{T}$ with top $k \in K$ premises from a ranking provided by Magnushammer, where $\mathcal{T}$ is a prescribed set of tactics and $K = \{1, 2, 4, 8, \ldots, 1024\}$. Such constructed proof steps are then executed in Isabelle. (See Algorithm 3 and Appendix C for details.)

In the single-step setting, Magnushammer outperforms Sledgehammer by a wide margin on both PISA (59.5% vs. 38.3%) and miniF2F (34.0% vs. 20.9%). Additionally, on PISA, Magnushammer outperforms TF-IDF and BM25: text-based, non-trainable retrieval methods [38] which are strong baselines in common retrieval benchmarks [43]. This suggests that Magnushammer is able to learn more than just superficial text similarity.

Interestingly, retrieval based on the generic OpenAI embeddings [27] (specifically: text-embedding-ada-002) yields reasonable performance comparable to Sledgehammer. This confirms the potential of neural premise selection to replace traditional symbolic methods. There is, however, a large gap to match Magnushammer. This shows that contrastive fine-tuning on our dataset provides non-trivial gains and supports our hypothesis that Magnushammer learns more than just mere textual similarity.

Multi-step setting Neural theorem provers often utilize language models to generate full proof steps, following the approach proposed in [35]. This allows for the creation of more complex, multi-step proofs. The proof generation involves sampling a proof step from the language model, verifying it, and repeating this process until the proof is closed or the computational budget is exceeded. The best-first search algorithm is often used to explore the most promising proof steps.

Thor [18] extends the capabilities of neural theorem provers by allowing them to generate proof steps utilizing an external premise selector – specifically, Sledgehammer. We modify Thor by replacing Sledgehammer with Magnushammer, which constituted Thor + Magnushammer architecture. (See

Appendix C.3 for details). Thor + Magnushammer establishes a new state of the art on the PISA benchmark (71.0% vs. 57.0%). On miniF2F, our method also significantly outperforms Thor and achieves results competitive with the current state of the art.

It is important to note that other theorem-proving approaches in the multi-step section of Table 2 require much larger language models: for Thor it is 700M non-embedding parameters; DSP (Draft, Sketch, and Prove) by Jiang et al. [19] uses Minerva model [25] with 62B parameters. Moreover, these other approaches rely on ideas orthogonal to premise selection. Specifically, Thor + auto [48] proposes a variation of Thor, involving expert iteration on auto-formalized data. DSP involves creating a high-level outline of a proof and uses Sledgehammer to solve the low-level subproblems. We hypothesize that both methods would perform even better when combined with Magnushammer.

5 Related work

Existing works on premise selection use classical machine learning like Bayesian and kernel methods [23, 1], k-NN [6], decision trees [33, 26, 34], and more recently, deep learning. Effective deep learning approaches often leverage the structure of mathematical expressions using graph neural networks [47, 30, 13]. Han et al. [14] use contrastive learning in informal premise selection. Concurrently to our work Yang et al. [50] develop a premise selection method for Lean similar to our SELECT method. Our work uses the transformer architecture [45], which is highly scalable and capable of producing powerful representations of text data. Unlike traditional hammers [32, 20, 12, 9], our method does not depend on external ATPs and requires little domain-specific knowledge.

Pre-trained transformer language models have been applied to various aspects of theorem proving, including tactic prediction [49], proof step search [35, 24], and autoformalization [48, 19]. The application of generative language models to premise selection has been limited, as the length of the possible premises often greatly exceeds the context of several thousand tokens that the models are designed to handle. Thor [18] circumvents the difficulty of premise selection by invoking Sledgehammer. In contrast, Magnushammer retrieves rather than generates to overcome the context length limitation. Therefore it can be used in tandem with other models (its combination with Thor is demonstrated in Section 4).

Batch-contrastive learning is widely used in speech [44], text [16], image [7] and image-text [37] representation learning. These methods have proven effective despite the possibility of false negatives occurring in contrastive batches [39]. The SELECT phase of our premise selection model relies on in-batch negative examples to train the retriever, similar to HOList [3] and Contriever [16]. Like HOList, we mine additional negatives, which we found crucial for performance. The RERANK stage closely resembles [29], but instead of using BM25, we jointly train retrieval and re-ranking, utilizing premises retrieved by SELECT as hard negatives for RERANK training.

6 Conclusion

In this paper, we introduced Magnushammer, a neural premise selection method that is transferable across proof assistants. We evaluate it in the Isabelle environment, showing that it outperforms the popular tool Sledgehammer on two benchmarks: PISA and miniF2F. Magnushammer can be plugged into automated reasoning systems as the premise selection component, as showcased with Thor. With its ease of adoption and high performance even with a low computational budget, Magnushammer paves the way for the firmer integration of deep-learning-powered tools into proof assistants.

References

[1] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. J. Autom. Reason., 52 (2):191–213, 2014. doi: 10.1007/s10817-013-9286-5. URL https://doi.org/10.1007/s10817-013-9286-5.

[2] Alexander A. Alemi, François Chollet, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath – deep sequence models for premise selection. CoRR, abs/1606.04442, 2016. URL http://arxiv.org/abs/1606.04442.

[3] Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of Proceedings of Machine Learning Research, pages 454–463. PMLR, 2019. URL `http://proceedings.mlr.press/v97/bansal19a.html`.

[4] Yves Bertot. A short presentation of Coq. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings, volume 5170 of Lecture Notes in Computer Science, pages 12–16. Springer, 2008. doi: 10.1007/978-3-540-71067-7_3. URL `https://doi.org/10.1007/978-3-540-71067-7_3`.

[5] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. J. Autom. Reason., 51(1):109–128, 2013. doi: 10.1007/s10817-013-9278-5. URL `https://doi.org/10.1007/s10817-013-9278-5`.

[6] Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for Isabelle/HOL. J. Autom. Reason., 57(3): 219–244, 2016. doi: 10.1007/s10817-016-9362-8. URL `https://doi.org/10.1007/s10817-016-9362-8`.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 1597–1607. PMLR, 2020. URL `http://proceedings.mlr.press/v119/chen20j.html`.

[8] Alexis Conneau and Guillaume Lample. Cross-lingual language model pretraining. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/c04c19c2c2474dbf5f7ac4372c5b9af1-Paper.pdf`.

[9] Lukasz Czajka and Cezary Kaliszyk. Hammer for Coq: Automation for dependent type theory. J. Autom. Reason., 61(1-4):423–453, 2018. doi: 10.1007/s10817-018-9458-4. URL `https://doi.org/10.1007/s10817-018-9458-4`.

[10] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, Automated Deduction – CADE-25 – 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings, volume 9195 of Lecture Notes in Computer Science, pages 378–388. Springer, 2015. doi: 10.1007/978-3-319-21401-6_26. URL `https://doi.org/10.1007/978-3-319-21401-6_26`.

[11] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB dataset of diverse text for language modeling. CoRR, abs/2101.00027, 2021. URL `https://arxiv.org/abs/2101.00027`.

[12] Thibault Gauthier and Cezary Kaliszyk. Premise selection and external provers for HOL4. In Xavier Leroy and Alwen Tiu, editors, Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, Mumbai, India, January 15-17, 2015, pages 49–57. ACM, 2015. doi: 10.1145/2676724.2693173. URL `https://doi.org/10.1145/2676724.2693173`.

[13] Zarathustra Amadeus Goertzel, Jan Jakubuv, Cezary Kaliszyk, Miroslav Olsák, Jelle Piepenbrock, and Josef Urban. The Isabelle ENIGMA. In June Andronick and Leonardo de Moura, editors, 13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel, volume 237 of LIPIcs, pages 16:1–16:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi: 10.4230/LIPIcs.ITP.2022.16.

[14] Jesse Michael Han, Tao Xu, Stanislas Polu, Arvind Neelakantan, and Alec Radford. Contrastive finetuning of generative language models for informal premise selection. 6th Conference on Artificial Intelligence and Theorem Proving, 2021.

[15] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In Iryna Gurevych and Yusuke Miyao, editors, Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers, pages 328–339. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1031. URL https://aclanthology.org/P18-1031/.

[16] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Towards unsupervised dense information retrieval with contrastive learning. CoRR, abs/2112.09118, 2021. URL https://arxiv.org/abs/2112.09118.

[17] Albert Q. Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. LISA: Language models of ISAbelle proofs. 6th Conference on Artificial Intelligence and Theorem Proving, 2021.

[18] 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. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=fUeOyt-2EOp.

[19] Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. CoRR, abs/2210.12283, 2022. doi: 10.48550/arXiv.2210.12283.

[20] Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. Math. Comput. Sci., 9(1):5–22, 2015. doi: 10.1007/s11786-014-0182-0. URL https://doi.org/10.1007/s11786-014-0182-0.

[21] Cezary Kaliszyk, François Chollet, and Christian Szegedy. HolStep: A machine learning dataset for higher-order logic theorem proving. CoRR, abs/1703.00426, 2017. URL http://arxiv.org/abs/1703.00426.

[22] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In Natasha Sharygina and Helmut Veith, editors, Computer Aided Verification – 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings, volume 8044 of Lecture Notes in Computer Science, pages 1–35. Springer, 2013. doi: 10.1007/978-3-642-39799-8_1. URL https://doi.org/10.1007/978-3-642-39799-8_1.

[23] Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, Automated Reasoning – 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings, volume 7364 of Lecture Notes in Computer Science, pages 378–392. Springer, 2012. doi: 10.1007/978-3-642-31365-3_30. URL https://doi.org/10.1007/978-3-642-31365-3_30.

[24] Guillaume Lample, Timothee Lacroix, Marie Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. HyperTree proof search for neural theorem proving. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=J4pX8Q8cxHH.

[25] Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=IFXTZERXdM7.

[26] Yutaka Nagashima and Yilun He. PaMpeR: Proof Method Recommendation system for Isabelle/HOL, 2018. URL https://arxiv.org/abs/1806.07239.

[27] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, Johannes Heidecke, Pranav Shyam, Boris Power, Tyna Eloundou Nekoul, Girish Sastry, Gretchen Krueger, David Schnurr, Felipe Petroski Such, Kenny Hsu, Madeleine Thompson, Tabarak Khan, Toki Sherbakov, Joanne Jang, Peter Welinder, and Lilian Weng. Text and code embeddings by contrastive pre-training, 2022.

[28] Tobias Nipkow. Fun with functions. Archive of Formal Proofs, August 2008. ISSN 2150-914x. https://isa-afp.org/entries/FunWithFunctions.html, Formal proof development.

[29] Rodrigo Frassetto Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. CoRR, abs/1901.04085, 2019. URL http://arxiv.org/abs/1901.04085.

[30] Aditya Paliwal, Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pages 2967–2974. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/5689.

[31] Lawrence C. Paulson. Isabelle: The next 700 theorem provers. CoRR, cs.LO/9301106, 1993. URL https://arxiv.org/abs/cs/9301106.

[32] Lawrence Charles Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In IWIL@LPAR, 2012.

[33] Bartosz Piotrowski and Josef Urban. ATPboost: Learning premise selection in binary setting with ATP feedback. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, Automated Reasoning – 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, volume 10900 of Lecture Notes in Computer Science, pages 566–574. Springer, 2018. doi: 10.1007/978-3-319-94205-6_37. URL https://doi.org/10.1007/978-3-319-94205-6_37.

[34] Bartosz Piotrowski, Ramon Fernández Mir, and Edward W. Ayers. Machine-learned premise selection for Lean. In Revantha Ramanayake and Josef Urban, editors, Automated Reasoning with Analytic Tableaux and Related Methods - 32nd International Conference, TABLEAUX 2023, Prague, Czech Republic, September 18-21, 2023, Proceedings, volume 14278 of Lecture Notes in Computer Science, pages 175–186. Springer, 2023. doi: 10.1007/978-3-031-43513-3_10. URL https://doi.org/10.1007/978-3-031-43513-3_10.

[35] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. CoRR, abs/2009.03393, 2020. URL https://arxiv.org/abs/2009.03393.

[36] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

[37] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. CoRR, abs/2103.00020, 2021. URL https://arxiv.org/abs/2103.00020.

[38] Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. Found. Trends Inf. Retr., 3(4):333–389, 2009. doi: 10.1561/1500000019.

[39] Joshua David Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021. URL https://openreview.net/forum?id=CR1XOQ0UTh-.

[40] Stephan Schulz. System description: E 0.81. In David Basin and Michaël Rusinowitch, editors, Automated Reasoning, pages 223–228, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-25984-8.

[41] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. CoRR, abs/2104.09864, 2021. URL https://arxiv.org/abs/2104.09864.

[42] Terence Tao. Solving mathematical problems: A personal perspective. Oxford University Press, 2010.

[43] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogenous benchmark for zero-shot evaluation of information retrieval models. CoRR, abs/2104.08663, 2021. URL https://arxiv.org/abs/2104.08663.

[44] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. CoRR, abs/1807.03748, 2018. URL http://arxiv.org/abs/1807.03748.

[45] 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. URL http://arxiv.org/abs/1706.03762.

[46] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

[47] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. CoRR, abs/1709.09994, 2017. URL http://arxiv.org/abs/1709.09994.

[48] Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus Norman Rabe, Charles E Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, Advances in Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=IUikebJ1Bf0.

[49] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of Proceedings of Machine Learning Research, pages 6984–6994. PMLR, 2019. URL http://proceedings.mlr.press/v97/yang19a.html.

[50] 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. CoRR, abs/2306.15626, 2023. doi: 10.48550/arXiv.2306.15626. URL https://doi.org/10.48550/arXiv.2306.15626.

[51] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. miniF2F: A cross-system benchmark for formal olympiad-level mathematics. In The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net, 2022. URL https://openreview.net/forum?id=9ZPegFuFTFv.

Appendix

A Isabelle environment

This section contains visual examples of proofs in Isabelle and provides some configuration details of the environment.

A.1 Visualization of the Isabelle environment

Figure A.1 shows an example theorem and its proof, as seen in Isabelle's most popular IDE, jEdit. The theorem comes from an entry to the Archive of Formal Proofs – *Fun With Functions* [28]. It states that any mapping f from the set of natural numbers to itself that satisfies $f(f(n)) < f(n+1)$ must be the identity function. The proof starts with a simple induction and then refines the result to arrive at the thesis. This problem was included in Terence Tao's booklet *Solving Mathematical Problems* [42].

```
theorem identity1: fixes f :: "nat ⇒ nat"
assumes fff: "⋀n. f(f(n)) < f(Suc(n))"
shows "f(n) = n"
proof -
  { fix m n have key: "n ≤ m ⟹ n ≤ f(m)"
    proof(induct n arbitrary: m)
      case 0 show ?case by simp
    next
      case (Suc n)
      hence "m ≠ 0" by simp
      then obtain k where [simp]: "m = Suc k" by (metis not0_implies_Suc)
      have "n ≤ f(k)" using Suc by simp
      hence "n ≤ f(f(k))" using Suc by simp
      also have "... < f(m)" using fff by simp
      finally show ?case by simp
    qed }
  hence "⋀n. n ≤ f(n)" by simp
  hence "⋀n. f(n) < f(Suc n)" by(metis fff order_le_less_trans)
  hence "f(n) < n+1" by (metis fff lift_Suc_mono_less_iff[of f] Suc_eq_plus1)
  with ‹n ≤ f(n)› show "f n = n" by arith
qed
```

Figure A.1: An example theorem in Isabelle. The statement is highlighted in the orange frame and the body of the proof is in the green frame. In this proof, most of the lines contain two consecutive steps: the first formulates a new proposition, and the second proves it. See a detailed analysis of the line 8 of the proof in Figure A.2 below.

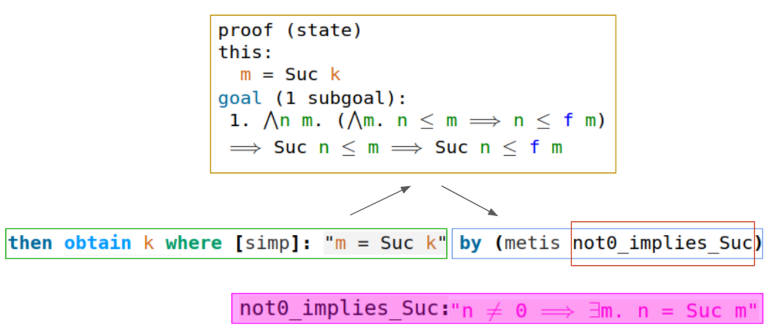


Figure A.2: The line is broken down into two steps: the first one (green frame) includes the proposition (since m is natural and positive, it must have a predecessor k) and the second (blue frame) proves it using the tactic `metis` with premise `not0_implies_Suc`, that states that a nonnegative natural number is a successor of some other natural number. The used premise is a fact which is already defined in the lemma library. The proof state resulting from the first step is in the yellow frame. The full premise statement is highlighted in pink.

A.2 Alternative proof step generation with Sledgehammer

This section describes how to generate alternative proof steps using Sledgehammer which we do to obtain datasets described in Section 3. First, we find all intermediate propositions within the proof (they can be nested) and try to replace the proof of the proposition with a Sledgehammer step. If successful, we record such a step in the dataset and proceed with both the original and the alternative proof. Figure A.3 provides a visual example of the aforementioned propositions.

```
proof -
  { fix m n have key: "n ≤ m ⟹ n ≤ f(m)"
    proof(induct n arbitrary: m)
      case 0 show ?case by simp
    next
      case (Suc n)
      hence "m ≠ 0" by simp
      then obtain k where [simp]: "m = Suc k" by (metis not0_implies_Suc)
      have "n ≤ f(k)" using Suc by simp
      hence "n ≤ f(f(k))" using Suc by simp
      also have "... < f(m)" using fff by simp
      finally show ?case by simp
    qed }
  hence "⋀n. n ≤ f(n)" by simp
  hence "⋀n. f(n) < f(Suc n)" by(metis fff order_le_less_trans)
  hence "f(n) < n+1" by (metis fff lift_Suc_mono_less_iff[of f] Suc_eq_plus1)
  with ‹n ≤ f(n)› show "f n = n" by arith
```

Figure A.3: Example intermediate propositions highlighted in red. Note: not all propositions were highlighted.

A.3 Example of a proof with tactics requiring premises

Figure A.4 contains a multi-step proof of the irrationality of $\sqrt{2}$ written in Isabelle. The proof contains multiple usages of tactics that require premises.

```
lemma "sqrt 2 ∉ ℚ"
proof
  assume "sqrt 2 ∈ ℚ"
  then obtain a b::int where "sqrt 2 = a/b" "coprime a b" "b ≠ 0"
    by (metis Rats_cases' less_irrefl)
  then have c: "2 = a^2 / b^2"
    by (smt (z3) of_int_power power_divide real_sqrt_pow2)
  then have "b^2 ≠ 0" by fastforce
  then have *: "2*b^2 = a^2"
    by (smt (verit, ccfv_SIG) c comm_semiring_class.distrib
        eq_divide_eq_numeral(1) mult_cancel_right1 numeral_Bit0
        numeral_plus_numeral of_int_add of_int_power
        of_int_power_eq_of_int_cancel_iff one_plus_numeral)
  then have "even a"
    by (smt (z3) even_power oddE)
  then obtain c::int where "a=2*c" by blast
  with * have "b^2 = 2*c^2" by auto
  then have "even b"
    by (smt (z3) even_power oddE)
  with ‹coprime a b› ‹even a› ‹even b› show False by fastforce
qed
```

Figure A.4: A proof of $\sqrt{2} \notin \mathbb{Q}$ [18, Figure 1]. The steps containing `metis, smt, fastforce, blast, auto, fastforce` are examples of steps using premises. For instance, one such proof step is `by (metis Rats_cases' less_irrefl)`. This step invokes `metis` and provides two premises as arguments, namely `Rats_cases'` and `less_irrefl`.

A.4 Sledgehammer setup

We set up Sledgehammer in Isabelle 2021-1, following the configuration used by [18]. We run Sledgehammer using different sets of settings and calculate the total proof rate by taking the union of problems solved by each run. The Sledgehammer timeout is set to default 30 seconds. We use only on-machine automated theorem provers (same as Isabelle environment), so external provers used by Sledgehammer are the following: Z3, SPASS, Vampire, CVC4, and E.

In our calculation of the Sledgehammer computation budget we assume $S = 10$ 'CPU cores.' We run our experiments on machines with 96 CPU cores, making the assumption realistic. Moreover, we emphasize that the performance gap between Magnushammer and Sledgehammer is large enough that altering the value of S, e.g., to an unrealistic level $S = 1$, would not qualitatively change conclusions.

B Details of Magnushammer training

We train Magnushammer in the two separate tasks alternating update steps as presented in Algorithm 1. Note that the backbone of the architecture is shared between SELECT and RERANK; such multi-task training is potentially more effective than having two separate models.

Algorithm 1 Magnushammer training.

```
Require:
    θ        ▷ initial trainable parameters
    D        ▷ premise dataset
    T        ▷ interval for updating rerank dataset
D_rerank ← recompute_negatives_for_rerank(θ, D)
step = 0
while step < num_train_steps do
    batch_select ← D.sample()
    θ ← train_step(θ, batch_select)
    batch_rerank ← D_rerank.sample()
    θ ← train_step(θ, batch_rerank)
    step ← step + 1
    if step mod T = 0 then
        D_rerank ← recompute_negatives_for_rerank(θ, D)
```

B.1 SELECT stage

SELECT stage is trained using the InfoNCE loss [44] defined as:

$$\mathcal{L}\left(q, k_{+}\right) = -\frac{\exp\left(s\left(q, k_{+}\right)/\tau\right)}{\exp\left(s\left(q, k_{+}\right)/\tau\right) + \sum_{i=1}^{K} \exp\left(s\left(q, k_{i}\right)/\tau\right)},$$

where q is a query (a proof state), k_{+} is a positive premise (a ground truth from the dataset), k_i are negative premises. We define s as cosine similarity between proof state and premise embeddings; $\tau > 0$ is a non-trainable temperature parameter.

Calculation of the negative premises for SELECT is costly, thus for efficiency reasons we recalculate the top 1024 premises every $T = 1000$ steps in the `recompute_negatives_for_rerank` function, as outlined in the Algorithm 1.

B.2 RERANK stage

Premise retrieval task can be cast as binary classification, trying to determine if a given pair $(\texttt{proof\_state}, \texttt{premise})$ is relevant. Applying classification to each pair is computationally infeasible, however, it could be used to *re-rank* a small set of premises retrieved by SELECT. Namely, we use the following cross-entropy loss:

$$\mathcal{L} = -\sum_{p \in \mathcal{P}} \log \texttt{score}(p) - \sum_{p \notin \mathcal{N}} \log(1 - \texttt{score}(p)),$$

where $\texttt{score}(p)$ is the output of the RERANK part of the model (see "Sigmoid" in Figure 1b) for a given $p = (\texttt{proof\_state}, \texttt{premise})$ pair. Typically, we sample a batch of 16 positive pairs $\mathcal{P}$ from the dataset. For each such pair $(\texttt{proof\_state}, \texttt{premise})$ 15 negatives are constructed from the most likely false positives returned by SELECT. Specifically, negative premises $\mathcal{M}$, which are facts that were never used as a premise for proof_state, are first chosen. Then, the top 1024 of $\mathcal{M}$ according to SELECT are selected, and 15 are sampled from them to construct negative pairs, which are included in $\mathcal{N}$.

B.3 Model architecture

We use a decoder-only transformer architecture, following the setup from [46] and using rotary position embedding by [41], a variation of relative positional encoding. The feedforward dimension in the transformer block is set to $4 \times D$ where D denotes embedding dimension, and the number of attention heads is $H = D/64$. Our 38M model has $L = 12$ layers and an embedding dimension of $D = 512$. The larger 86M model consists of $L = 12$ layers and has $D = 768$. For all the models, we use the original GPT-2 tokenizer [36]. The results presented in the main body of the paper were obtained using the larger, 86M-parameter model.

In SELECT, we append a specialized token at the end of the sequence to compute the embedding for a proof state and linearly project its embedding. Premises are embedded analogously. Similarly to [37] that train separate projections for images and captions, we train separate proof state and premise projections and share the transformer backbone (see Figure 1b). Analogously for RERANK, we compute the relevance score by taking the embedding of the last token and then projecting it to a scalar value.

B.4 Hyperparameter setup

We performed the following hyperparameter sweeps. We note that we have not observed significant differences between obtained results.

- Learning rate: $\{1\text{e}{-4}, 2\text{e}{-4}, 3\text{e}{-4}, 5\text{e}{-4}\}$, chosen: $2\text{e}{-4}$
- Dropout: $\{0.0, 0.05, 0.1, 0.2\}$, chosen: 0.1
- Weight decay: $\{0.02, 0.05, 0.1\}$, chosen: 0.02
- Batch size N in SELECT: $\{128, 256, 512\}$, chosen: 256
- Number of negatives M in SELECT: $\{0, 256, 768, 1536\}$, chosen: 768
- Temperature for InfoNCE loss in SELECT: $\{0.05, 0.07, 0.2, 1\}$, chosen: 0.07
- Batch size for RERANK: $\{16, 32, 64\}$, chosen 64
- Number of negatives per proof state $\mathcal{M}$ in RERANK: $\{7, 15\}$, chosen: 15.

B.5 Pre-training on language modeling

Pre-training has been shown to dramatically increase the capabilities and performance of decoder-only models on tasks other than language modeling [15]. Motivated by that, we pre-train our models on GitHub and arXiv subsets of the Pile [11].[4] The models are trained for 1M steps, with a context length of 2048. Global batch size is set to 32 sequences giving a total number of 65536 tokens per batch. Dropout is disabled, and weight decay is set to 0.02. The learning rate increases linearly from 0 to 0.0003 for the first 10000 steps, and then the cosine schedule is applied to decrease its value gradually.

B.6 Fine-tuning for downstream tasks

We train Magnushammer by taking a pre-trained language model, removing its language modeling head, and attaching three linear projection heads – one projection for proof state embedding, another

[4]We follow here the methodology of Polu and Sutskever [35] who verified that generative pre-training substantially improves proving performance in MetaMath and that pre-training on mathematical data leads to better performance compared to pre-training on generic text from the web.

one for premise embedding, and the last one for producing relevance score for RERANK, as depicted in Figure 1b and described in Section B.3. For the proof step generation task, we fine-tune our language models by applying the algorithm used to train Thor [18].

B.7 Hardware

We gratefully acknowledge that our research was supported with Cloud TPUs from Google's TPU Research Cloud (TRC). We use TPU virtual machines from the Google Cloud Platform (GCP) for all stages: pre-training, fine-tuning, and evaluation. Each TPU virtual machine has 8 TPU v3 cores, 96 CPU cores, and over 300GB of RAM. TPU v3 cores have around 16GB of memory each. The Isabelle environment is set to have access to 32 CPU cores.

C Details of Magnushammer evaluation

Algorithm 2 shows the two-stage premise selection method of Magnushammer.

Algorithm 3 outlines the evaluation method described in Section 4. To generate the proof steps there, we use the following tactics: `smt`, `metis`, `auto`, `simp`, `blast`, `meson`, `force`, `eval`, `presburger`, `linarith`.

Algorithm 2 Premise selection with Magnushammer.

```
Require:
    proof_state                                   ▷ proof state to retrieve premises for
    premises                                      ▷ database of available premises
    K_S, K_R      ▷ number of premises to retrieve with SELECT and RERANK, respectively
state_embedding ← get_embeddings(proof_state)                 ▷ SELECT stage starts
premises_embeddings ← get_embeddings(premises)
Cache(premises_embeddings)
sim_scores = state_embedding · premises_embeddings
selected = premises[argsort(−sim_scores)[: K_S]]
batch = []                                                    ▷ RERANK stage starts
for premise in selected do
    batch.append((proof_state, premise))
rerank_scores ← get_rerank_scores(batch)
top_premises = selected[argsort(−rerank_scores)[: K_R]]
return top_premises
```

Algorithm 3 Magnushammer evaluation in ITP environment.

```
Require:
    theorem                                                   ▷ theorem to prove
    premsel_model                        ▷ Magnushammer's premise selection model
    K_S                                  ▷ number of premises to retrieve with SELECT
    K_R                                  ▷ number of premises to retrieve with RERANK
    premises                                                  ▷ available premises
    top_k_premises_to_try   ▷ list with the number of top premises to generate steps with
    tactics_to_try                          ▷ list of tactics to generate steps with
    env                                        ▷ ITP environment (e.g., Isabelle)
proof_state ← init_problem(env, theorem)                      ▷ initialize problem
top_premises ← premsel_model(proof_state, premises, K_S, K_R)  ▷ get top premises
steps = []               ▷ generate proof steps combining of tactics and top k premises
for k in top_k_premises_to_try do
    top_k_premises ← top_premises[: k]
    new_steps ← generate_steps(tactics_to_try, top_k_premises)
    steps.extend(new_steps)
solved ← try_steps(env, steps) ▷ evaluate generated proof steps in the ITP's environment
return solved
```

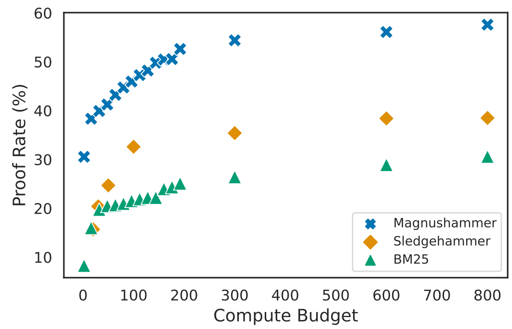


Figure C.5: Proof success rate for varying computational budget for Magnushammer, Sledgehammer, and BM25. Magnushammer shows remarkable scalability.

C.1 Computational budget

For our main results (Section 4), we allocate the computational budget of 1000 as follows: apart from the powers of two from 2^0 to 2^{10}, we also try the following k values: $[48, 96, 192]$, which in total gives 14 values. With each of these k values, 36 tactics are used with timeout $T = 2$, yielding $C \approx 1000$.

For the ablation studies, we only use powers of two from 2^0 to 2^{10}, and the same set of 36 tactics, which gives $C \approx 800$.

C.2 Scaling computational budget

Figure C.5 shows how the quality of premise selection methods varies with the computational budget available during evaluation. Notably, Magnushammer outperforms Sledgehammer even with very limited computational resources, and it scales well, particularly within the medium budget range.

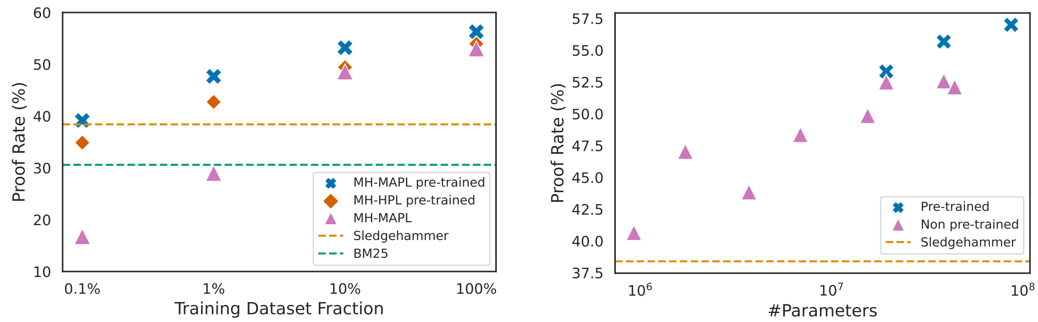
For Magnushammer and BM25, we use Algorithm 3 in various configurations (i.e., settings of $\mathcal{T}$ and K). We start with one tactic, $\mathcal{T} = \{\texttt{smt}\}$, and $K = [2^7]$, which yields $C = 2$ (recall that $T = 2$ s). We then gradually add more tactics to $\mathcal{T}$ and more values to K. The final setup uses $|\mathcal{T}| = 36$ and K containing all powers of 2, from 2^0 up to 2^{10}, which yields $C \approx 800$. Details are provided in Appendix C. For Sledgehammer, we scale the timeout parameter T up to 80 s. We use models trained on the MAPL dataset and evaluate them with a computational budget of 800.

C.3 Thor + Magnushammer

To generate more complex proofs we combine Thor [18] with Magnushammer as introduced in multi-step setting in Section 4.

Firstly, we follow the procedure described in [18] to pre-process training data and fine-tune our pre-trained language model for the proof generation task (pre-training details can be found in Appendix B.5). During the evaluation, when the language model generates the `<hammer>` token, we call our method instead of Sledgehammer. More specifically, we use an augmented Algorithm 3 that returns the proof states resulting from applying the steps (instead of returning binary information on whether any of the steps closed the proof). We then pick at most s = 2 states among these and add them to the BFS queue.

We assign the same computational budget as proposed in Thor, with the only difference being that each `proof_step` has a timeout limit of 2 s (instead of 10 s), which we found to perform better in our setup. The search is terminated if and only if one of the following scenarios happens: (1) a valid proof has been found for the theorem; (2) the language model is queried 300 times; (3) a wall-time timeout of 500 s has been reached (assuming parallel execution of Magnushammer steps); (4) the queue is empty but the theorem is not proved. We keep the same maximum length of the queue equal to 32.



(a) We randomly sample fractions of MAPL or HPL datasets and use them for training Magnushammer. Even 0.1% of the MAPL dataset allows pre-trained Magnushammer to outperform the Sledgehammer and BM25 baselines.

(b) We train Magnushammer of different sizes. Even with a one-layer transformer, Magnushammer outperforms Sledgehammer. We observe consistent performance gains with increasing model sizes. Pre-trained models perform better.

Figure D.6: Impacts of the training data quantity and the model parameters on the proof rate. The vertical axis is the proof rate in percentage. In Subfigure D.6a, the horizontal axis is the fraction of training dataset used and in Subfigure D.6b it is the number of parameters in the model.

D Ablations

D.1 Impact of training data

We study how the amount and type of data impact the proof success rate by comparing HPL and MAPL datasets. For this comparison, we used models with 38M non-embedding parameters and a computational budget of 800.

Dataset size Our method is data-efficient: see Figure D.6a. We observe that Magnushammer fine-tuned on only 0.1% of MAPL – equivalent to approximately 4K samples – is already able to outperform Sledgehammer. This indicates that when starting from a pre-trained model, Magnushammer is a promising approach for addressing premise selection in theorem-proving environments with limited training data. The effect of pre-training diminishes as the amount of training data increases.

Dataset type Fine-tuning on MAPL or HPL leads to subtle differences (56.3% vs. 54.0% when the whole datasets are used). This outcome may be attributed to the impact of model pre-training and the fact that the HPL dataset is rich enough to obtain good performance on the PISA benchmark (as observed in the previous paragraph). We speculate that the bigger MAPL dataset might be essential for future harder benchmarks and scaling up the model size.

Model size To study how the performance of our method depends on the model size, we vary the number of layers L and embedding dimension D. A positive correlation between the model size and the proof rate is shown in Figure D.6b. We observe that even a tiny model with 920K parameters ($L = 1, D = 256$) outperforms Sledgehammer (40.7% vs. 38.3%). We also note the benefit of pre-training and that scaling the number of layers is more beneficial than scaling the embedding dimension. Details, including the configuration of each model, are in Appendix B.3.

Impact of re-ranking We find that the SELECT-only method, i.e., Magnushammer without the RERANK phase, already significantly outperforms Sledgehammer. Tested on the 38M model, it achieves a 54.2% proof rate comparable to 56.3% obtained by Magnushammer. SELECT-only mode is a computationally appealing alternative, as it only needs a single forward pass to embed the current proof state (the setting used recently by Yang et al. [50].) Premise embeddings can be pre-computed and cached, allowing inference on the CPU without the need for GPU or TPU accelerators.