Temperature-scaled large language models for Lean proofstep prediction

Fabian Gloeckle
FAIR at Meta
CERMICS École des Ponts ParisTech

Baptiste Rozière
FAIR at Meta

Amaury Hayat
CERMICS École des Ponts ParisTech

Gabriel Synnaeve
FAIR at Meta

Abstract

Leveraging the reasoning capabilities of large language models (LLMs) for theorem proving is a promising but challenging task, requiring in-domain finetunings on which LLMs are prone to overfit. This issue is exacerbated by two properties that set theorem proving apart from more mainstream applications of LLMs: the scarcity of training data in formal environments such as Lean or Isabelle and the prohibitive cost of using evaluation benchmarks for hyperparameter search and model selection. In this work, we propose *temperature scaling* as a regularization method for multi-epoch training on small datasets. We demonstrate its effectiveness empirically, obtaining state-of-the-art supervised tactic generation models for Lean 3 of sizes 1.5B, 7B and 13B parameters. We provide detailed ablations on proof search hyperparameters and analyses of the proof search behaviors of the resulting models, and exhibit the approximate logarithmic scaling of tactic-based proof search with respect to time budget.

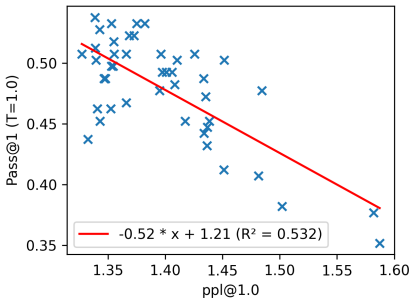
1 Introduction

With the increasing availability of large language models following the release of Llama [Touvron et al., 2023a], new avenues have opened up for research in neural theorem proving. While having been employed in approaches based on natural language [Lewkowycz et al., 2022] [Jiang et al., 2023] [Wu et al., 2022] and zero-shot whole proof generation in formal environments [First et al., 2023], large language models have so far been conspicuously absent in tree-search based techniques in formal environments, where interactive theorem provers ground machine learning models and tree search provides an efficient method of exploration.

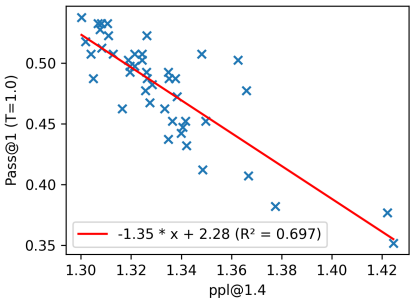
However, adapting large language models for proofstep prediction in formal environments comes with unique challenges. *(i)* Model finetuning is required because proofstep prediction is not easily few-shot learnable and not part of pretraining curricula. *(ii)* Training data in the form of human-written proofs in interactive theorem provers is scarce and large language models are prone to overfit when finetuned on small datasets. *(iii)* Downstream performance evaluations of proofstep prediction models are prohibitively costly for extensive hyperparameter searches. This third property sets formal theorem proving apart from other domains of large language model finetunings. It prevents the use of downstream metrics for extensive hyperparameter selection or early stopping, and calls for its own unique solutions.

Validation perplexity is a metric frequently used in both language model pretrainings and finetunings. However, in our experiments, we find that when applied naively in the context of theorem proving, perplexity leads to a suboptimal choice of model checkpoints.

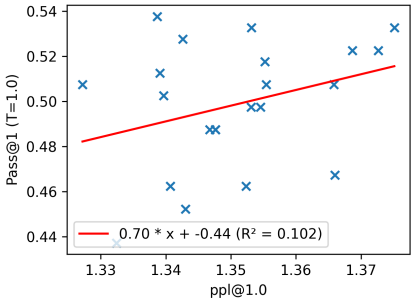
37th Conference on Neural Information Processing Systems (NeurIPS 2023) Workshop on MATH-AI.



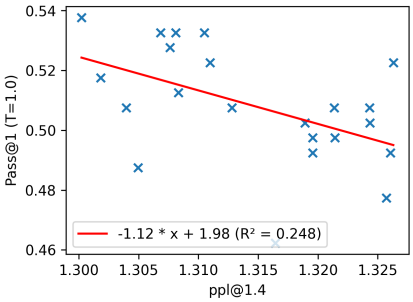
(a) All checkpoints



(b) All checkpoints



(c) 22 best checkpoints according to ppl@1.0



(d) 22 best checkpoints according to ppl@1.4

Figure 1: **Proof success rate (pass@1) by temperature-scaled perplexity (ppl@T).** 1a According to conventional theory, overfitting can be measured by validation perplexity (ppl@1.0). 1b Our experiments indicate, however, that evaluating the perplexity at a higher temperature (1.4 in this case, ppl@1.4) provides a better predictor for model quality in proving benchmarks. 1c Selecting only the 22 points with smallest validation perplexity – the ones considered for model selection in practice – ppl@1.0 is only weakly correlated with downstream performance and gives a signal in the opposite direction of what conventional theory predicts: checkpoints with *higher* ppl@1.0 perform better. 1d Perplexity evaluated at higher temperature stays correlated with downstream performance also in the top left corner of best model checkpoints, allowing for accurate model selection from a metric that is cheap to evaluate. Selecting model checkpoints based on this metric amounts to choosing later checkpoints classically dubbed as "overfitted" and improves scores in evaluations. Shown are a total of 45 evaluations of mid-training checkpoints from five independent runs of 1.5B parameter models, evaluated on the proof search benchmark with a sampling temperature of 1.0. The explained covariance R^2 of the depicted ordinary least squares regression increases from 0.532 for ppl@1.0 to 0.697 when switching to ppl@1.4.

In this work, we propose to use *temperature-scaled perplexity* [Guo et al., 2017] as an alternative method for model selection. We demonstrate empirically that it is more closely correlated with downstream metrics in theorem proving (Section 3), and discuss the reasons for this phenomenon in Appendix 10, where we link it to training error decompositions and provide an analogy to Gaussian mixture models.

Our contributions can be summarized as follows:

- We identify temperature scaling [Guo et al., 2017] as an effective method for model selection when finetuning large language models. Using it, performance improves considerably compared to using long-established unscaled perplexity (Figure 1 and Section 3).
- We finetune Code Llama models with 1.5B, 7B and 13B parameters, providing state-of-the-art proofstep predictors for Lean 3 outperforming the baseline by more than 10 %.
- We perform extensive experiments comparing model sizes and hyperparameters. We exhibit the *speed vs. accuracy tradeoff* for proofstep prediction models as well as the *logarithmic scaling* behavior of tactic-based proof search (Section 3).

Table 1: **Proof success rate (pass@1) of our models on the LeanDojo "random" benchmark.**

Method	Size	Pass@1 (%) (600 s timeout)	Pass@1 (%) (1200 s timeout)
`tidy`		23.8	23.8
ReProver	299M	51.4	
ReProver (w/o retrieval)	299M	47.5	
finetuned Code Llama	1.5B	56.2	58.0
finetuned Code Llama	7B	57.7	59.9
finetuned Code Llama	13B	51.5	55.7

2 Method

We finetune Code Llama [Rozière et al., 2023] models of sizes 1.5B, 7B and 13B parameters on a supervised dataset of proof state – tactic prediction pairs extracted from Lean 3's mathlib. According to preliminary experiments, code-trained Code Llama models provide a better starting point than natural language-trained Llama 2 models (Table 3). The models use the Llama tokenizer and a sequence length of up to 4096 tokens. Further details on the training hyperparameters as well as on the LeanDojo [Yang et al., 2023] dataset and evaluation benchmark used can be found in Appendix 6.

Scaling models and search We scale tactic-based proof search in two dimensions: in the number of model parameters and in the search budget allocated to each proof. Concerning *model size*, we expect a quality vs. quantity tradeoff between small models generating tactics of lower quality fast and large models generating tactics of higher quality at a slower pace. In order to produce a meaningful conclusion on this tradeoff, we need to ensure that we evaluate all models in their respective optimal settings, so we conclude an extensive hyperparameter study on parameters for tactic-based proof search. By increasing the *time budget* allocated to each proof, we examine the empirical distribution of proof difficulties and hope to lay the foundations for accurate extrapolation of proof metrics in future studies.

Model selection via temperature scaling Model finetuning on small datasets risks overfitting i.e. a degradation of downstream performance on later model checkpoints. This can be prevented by applying early stopping Goodfellow et al. [2016] with respect to downstream evaluation metrics or training metrics computed on a held-out validation dataset. Unlike in other domains of natural language processing, theorem proving in formal environments faces the challenge that evaluation metrics are expensive to compute: indeed, evaluations can exceed the cost of model finetuning *by two orders of magnitude!* It is therefore a natural choice to consider validation loss as a metric for early stopping. We have investigated its usefulness and observed its shortcomings for model selection in theorem proving (Section 3). Instead, we suggest to use *temperature-scaled perplexity* (explained below) as a metric for early stopping. In Section 3, we present empirical evidence underlining its usefulness for model selection in theorem proving, and in Section 4, we give heuristical theoretical explanations for the observed phenomena.

We propose to use *temperature-scaled perplexity* as a metric to determine the optimal early stopping point for multi-epoch finetunings on small datasets, with a dataset-dependent temperature $T > 1$. Here temperature scaling refers to the practice of smoothening a categorical distribution by dividing the model's output logits by a fixed constant before applying the softmax function [Hinton et al., 2014]. This technique has been suggested to improve the calibration of vision models *post hoc* [Guo et al., 2017], and we believe it can be useful to measure and reduce overfitting beyond the realm of classification tasks. We write ppl@T for perplexity based on the temperature-scaled distribution with temperature T, and consequently ppl@1.0 to denote classical validation perplexity.

3 Results

Our 7B parameter model reaches a proof rate of 57.7 % on the LeanDojo theorem proving benchmark without premise retrieval, up from 51.4 % for the retrieval-augmented baseline from Yang et al. [2023]. We detail our observations on optimal search hyperparameters, the differences in proving

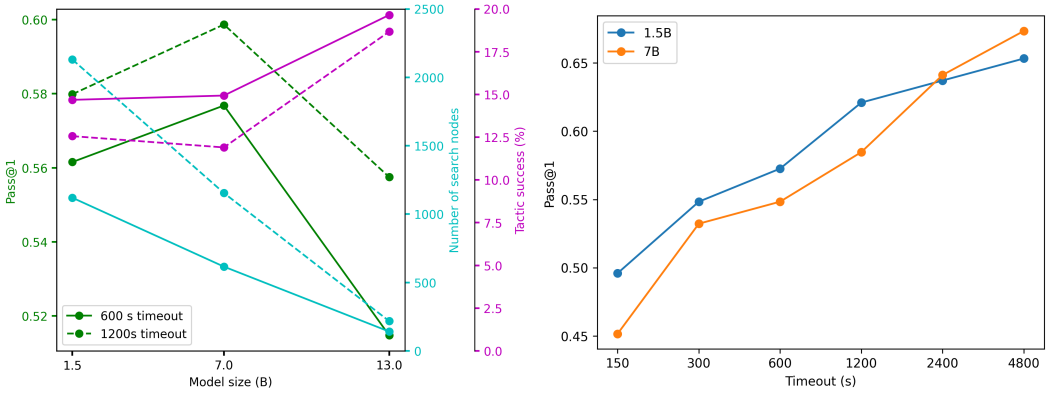


Figure 2: **Scaling model size and search budget. Left:** The optimal model size is subject to a quality vs. quantity tradeoff. Smaller models are able to explore more search nodes within the fixed time budget, while larger models have lower error rates on the generated tactics. The overall best proving performance is reached by the model of intermediate size with 7B parameters. Increasing the search budget has a similar effect across model sizes and affects both correlate metrics for quantity and quality as the distribution of encountered proof states changes. For further metrics exemplifying this tradeoff, see Table 5. **Right:** Pass@1 scores evaluated on 256 random test samples from the LeanDojo "random" benchmark in dependency of the timeout imposed on each individual proof. The logarithmic scaling aligns with findings on code generation benchmarks [Li et al., 2022a], as well as with the combinatorial nature of proof search.

behaviors of differently-sized models, logarithmic scaling with respect to search budget and optimal checkpoint selection via temperature-scaled perplexity below.

Hyperparameters for tactic-based proof search Before comparing models of different sizes on the theorem proving benchmark, we extensively benchmark proof search hyperparameters to evaluate each model at its respective optimal settings. Evaluating on a 200 element subset of the provided validation dataset, we find temperature 1.3 to be a good choice for the models with 1.5B and 7B parameters, and temperature 1.1 for the model with 13B parameters (Figure 4). These temperatures are larger than the optimal temperatures for code generation models of comparable sizes [Chen et al., 2021], stressing the combinatorial nature of proof search and the importance of case-adapted sampling parameters. We use nucleus sampling with a probability mass of 0.95. Increasing the temperature further and correcting by decreasing the nucleus probability mass does not improve the scores (Figure 5).

Tactic-based proof search needs to strike a balance between search breadth as expressed by the number of tactics evaluated per search node, and the search depth reachable within the search time. This tradeoff is linked with the exploration-exploitation tradeoff defined by the temperature at which tactic models are evaluated, so we evaluate a grid search on the number of samples per node and temperature for the 1.5B and 7B parameter models (Figure 6). The 7B parameter model performs best at 64 samples per node across temperatures, while the 1.5B parameter model has a less clearly defined maximum in the range from 64 to 384 samples per node.

Proving behavior by model size Table 1 shows the performance of our models on the LeanDojo "random" theorem proving benchmark for Lean 3. In addition, we evaluate our models on a range of metrics with the aim of discerning different qualitative behaviors of proof search at different model sizes. The metrics on which we focus can be categorized as end-to-end proving metrics, tactic generation quality metrics and tactic diversity metrics and are described in Appendix 11.

Tree-search based proving systems have to strike a balance between exploration through fast tactic generation and high tactic diversity and exploitation of known patterns through tactic quality. We observe qualitatively different behaviors of the different model sizes evaluated at their respective optimal temperatures. Figure 2 and Table 5 show our main results. Our findings match the following intuitive expectations.

- There exists a model size that optimizes the speed / accuracy trade-off in neural search-based theorem proving. It is 7B parameters for our models and data.
- Larger models produce higher-quality tactic suggestions and perform less search, as evidenced by their higher tactic success rate, shorter proofs and smaller numbers of searched nodes.
- Smaller models are implicitly regularized by their size and most useful at higher temperatures, leading to more diverse tactic suggestions.

Scaling search budget Increasing the per-proof timeout on proof searches from 150 seconds up to 4800 seconds, we exhibit an approximate logarithmic scaling behavior for proof rates (Figure 2). Note that the derivative describes the "density of proving problems per difficulty" and has the form of a power law.

Temperature scaling Our observations on checkpoint selection with models of size 1.5B parameters can be summarized as follows:

- Early stopping with respect to perplexity is suboptimal: performance on the proving benchmark peaks at a later point in finetuning and coincides with the minimum of *temperature-scaled perplexity* for a temperature $T > 1$ (Figure 7).
- More precisely, ppl@1.4 is correlated more closely with proof success rate than ppl@1.0, with a coefficient of explained covariance of $R^2 = 0.697$ instead of $R^2 = 0.532$ for ppl@1.0 (Figure 1).
- This appears not to be a fluctuation but a continuous phenomenon: R^2 increases monotonically up to a temperature of 1.4 before breaking down at higher temperatures (Figure 8). Investigating more closely, it is especially on the checkpoints with lowest ppl@T that the difference in regression fits appears, with the direction of the correlation even reversing for $T \leq 1.0$ (Figures 1c, 9).
- Higher temperatures are not only more predictive for proof success, but also minimize ppl@T across temperatures, with a minimum occurring likewise at $T = 1.4$ (Figure 10).

4 Discussion

Neural theorem proving in formal environments eventually needs to move toward more sophisticated techniques than pure tactic-based best-first search using model probabilities or AlphaGo-style Monte Carlo tree search [Silver et al., 2017, Lample et al., 2022] in order to overcome the logarithmic scaling behavior we exhibited (Figure 2). However, we believe that search-based techniques will continue to play a role in more hierarchical systems [Jiang et al., 2023], both as low-level black-box provers and as an adaptable and efficient method for exploration. For such, pretrained large language models (LLMs) offer access to useful priors, knowledge and general reasoning capabilities that could not be extracted from the limited data available in formal proof archives alone. At the same time, adapting LLMs for tactic-based proof search comes with unique challenges. Downstream performance metrics are not only expensive to compute but also depend strongly on the hyperparameters selected for the search process. In this study, we proposed temperature-scaled perplexity as a simple metric that improves model selection compared to standard perplexity while being considerably cheaper to evaluate than performance on benchmarks. Moreover, we investigated the tradeoffs of quality vs. speed of tactic generations defined by model size, of breadth vs. depth of proof search defined by the number of tactic samples per search node and of quality vs. diversity of generated tactics defined by sampling temperature. Facilitating the choice of search hyperparameters and model size, we hope to contribute to the democratization of LLM usage in theorem proving research and to the development of said hierarchical systems in the future.

More broadly, the problem of expensive downstream evaluations is not exclusive to theorem proving alone. In many real-world cases of model finetuning, no reliable benchmark is available, causing researchers to rely on expensive LLM or human annotations. Our proposed method of model selection by temperature-scaled perplexity could prove useful in such cases beyond the realm of theorem proving as a cheap and efficient proxy for downstream model performance. We leave the development and empirical verification of a recipe for temperature-scaling – the choice of temperature – that works well across domains for future work.

Acknowledgments and Disclosure of Funding

We especially want to thank David Lopez-Paz for helpful discussions and for bringing up the notion of calibration in the context of temperature scaling. Thanks also to Timothée Lacroix and Guillaume Lample for introducing the first author to the field of neural theorem proving and for sharing their expertise in machine learning theory and practice. Thanks to Albert Jiang for stimulating discussions on automated theorem proving throughout.

References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harri Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Raj Dabre and Atsushi Fujita. Softmax tempering for training neural machine translation models, 2020.

Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover. In *Automated Deduction - CADE-25, 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, 2015.

Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models, 2023.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks, 2017.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*, 2014.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2020.

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, 2023.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. Hypertree proof search for neural theorem proving, 2022.

Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, and Daniel Haziza. xformers: A modular and hackable transformer modelling library. `https://github.com/facebookresearch/xformers`, 2022.

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. *arXiv preprint arXiv:2206.14858*, 2022.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022a.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022b.

Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences, 2018.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, jan 2020. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145%2F3372885.3373824.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving, 2020.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning, 2022.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022.

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

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2023.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

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

Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. To repeat or not to repeat: Insights from scaling llm under token-crisis, 2023.

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.

Supplementary Material

5 Related work

Large language models for theorem proving. Large languages models have been used for solving mathematical problems in natural language [Lewkowycz et al., 2022], for creating proof sketches [Jiang et al., 2023] and autoformalization [Jiang et al., 2023] [Wu et al., 2022] or for zero-shot whole proof generation and repair [First et al., 2023]. Tree-search based proof techniques, on the other hand, have so far focussed on models of small size by today's standards, with parameter sizes ranging from 299M parameters [Yang et al., 2023] to 600M parameters [Lample et al., 2022] and 774M parameters [Polu and Sutskever, 2020] [Polu et al., 2022]. We believe that tree-search-based techniques will continue to play an important role in neural theorem proving as they represent a compelling method for exploration and a useful component of hierarchical theorem proving systems. To the best of our knowledge, this is the first work to study the impact of model scale on end-to-end theorem proving performance and to provide recommendations on optimal model size.

Temperature-based methods in the low data regime. While large language model pretrainings use one to two epochs per dataset [Touvron et al., 2023a] [Touvron et al., 2023b] [Chowdhery et al., 2022], model finetunings operate in the more classical data-constrained regime where the optimal point between underfitting and overfitting models is sought and multi-epoch training is beneficial. While it has been frequently noted that downstream metrics continue to improve beyond the onset of overfitting as measured by validation loss [Ouyang et al., 2022] [Power et al., 2022], we are not aware of attempts to quantify this observation and to suggest a procedure to decide on the number of epochs to use in multi-epoch training. The method of temperature-scaling has first been suggested in the context of calibration of computer vision models [Guo et al., 2017], and in this work we attempt to extend this from a post-hoc recalibration method to a general framework for understanding and handling concentration of measure in multi-epoch training. Softmax tempering [Hinton et al., 2014] Dabre and Fujita [2020] Li et al. [2022b] has been suggested as an alternative technique to make temperature-based metrics available for model finetunings, but it requires to decide on one desired temperature instead of being able to monitor a family of metrics over the course of training and decide based on the aggregated information obtained, and it remains unclear whether its implicit rescaling of the output layer's weights between pretraining and finetuning is beneficial or harmful.

6 Details on training hyperparameters, dataset and evaluation

Training For all finetunings, we use the Adam optimizer [Kingma and Ba, 2015] with $\beta_1 = 0.9$, $\beta_2 = 0.95$ and a decoupled weight decay coefficient [Loshchilov and Hutter, 2019] of 0.1. We warm up the learning rate linearly and decay it with cosine decay [Loshchilov and Hutter, 2017] over a fixed number of decay steps to a certain fraction of the peak learning rate. Table 2 details the hyperparameter choices for the respective model finetunings.

Table 2: **Hyperparameters for model finetunings.**

Model size	Batch size (tokens)	Learning rate	Decay ratio	Warmup steps	Decay steps	Checkpoint step
1.5B	128k	1.0×10^{-4}	0.1	200	4000	2600
7B	48k	5.5×10^{-5}	0.3	200	6000	4400
13B	48k	3.0×10^{-5}	0.1	200	6000	3800

For the 1.5B parameter variant, we use a model that follows Llama 2's [Touvron et al., 2023b] and Code Llama's [Rozière et al., 2023] training recipes with the architecture of GPT-2 [Radford et al., 2019].

Dataset LeanDojo [Yang et al., 2023] provides datasets consisting of proof states, tactic steps and metadata automatically extracted from the proofs in Lean 3's [de Moura et al., 2015] mathlib [mathlib Community, 2020]. As we are training models without retrieval mechanism, we focus on the provided "random" split, in which theorems are randomly assigned to the training, validation or test dataset. Overall, raining dataset consists of 204k proofstep samples, amounting to 201M characters or 104M tokens with the Llama tokenizer, including an outcome cotraining task.

Data formatting As our models follow a decoder-only [Liu et al., 2018] transformer architecture [Vaswani et al., 2017], we train on the (input, output) pairs autoregressively with prompts formatted as indicated in Figure 3. We include an outcome prediction cotraining task by appending the tactic outcome to the sample.

```
NAME: is_adic_complete.le_jacobson_bot FILE: src/linear_algebra/adic_completion.lean
STATEMENT: lemma le_jacobson_bot [is_adic_complete I R] : I ≤ (⊥ : ideal R).jacobson
STATE: R : Type u_1,
_inst_1 : comm_ring R,
I : ideal R,
_inst_6 : is_adic_complete I R,
x : R,
hx : x ∈ I
⊢ x ∈ ⊥.jacobson
TACTIC: rw [← ideal.neg_mem_iff, ideal.mem_jacobson_bot]
RESULT: R : Type u_1,
_inst_1 : comm_ring R,
I : ideal R,
_inst_6 : is_adic_complete I R,
x : R,
hx : x ∈ I
⊢ ∀ (y : R), is_unit (-x * y + 1)
```

Figure 3: **Example of the training data formatting used.** Note that we actually use spaces to separate the different fields as they are more robustly handled by the Llama tokenizer than newline characters.

Evaluation We evaluate our models on LeanDojo's proof search benchmark ("random" split) consisting of 2000 theorems from Lean 3's mathlib, using the best-first tree search code provided by the authors [1]. Following their evaluation procedure, we generate 64 tactics per node and impose

[1] `https://github.com/lean-dojo/ReProver`

a wall clock time limit of 10 minutes on each individual proof search unless otherwise specified. Ablations and hyperparameter searches are performed on a subsample of 200 theorems from the provided validation set. Where indicated, we use a random subset of 256 theorems from the test set instead of the full one. According to our experiments, this subset appears to be easier on average than the full test set, so that numbers computed on the smaller test set are not directly comparable to those obtained on the full one.

Adaptations to the evaluation framework We make the following adaptations to LeanDojo's evaluation framework in order to speed up evaluation. We reuse mathlib environments instead of creating a fresh one for each proof, which significantly reduces disk I/O time. Moreover, we use up to five Lean processes per tactic generator to execute proof steps. For tactic generation, we use dynamic batch sizes depending on the prompt length and efficient inference code that uses the xFormers transformers toolbox [Lefaudeux et al., 2022].

The benchmarking framework would benefit a lot from implementing goal splitting [Lample et al., 2022] which treats each goal [2] in a given proof state as a separate proof search (instead of making the entire list of current goals a part of the proof state), but this is beyond the scope of this work.

[2] more precisely: each metavariable-connected component

7 Hyperparameters for proof search

7.1 Sampling parameters

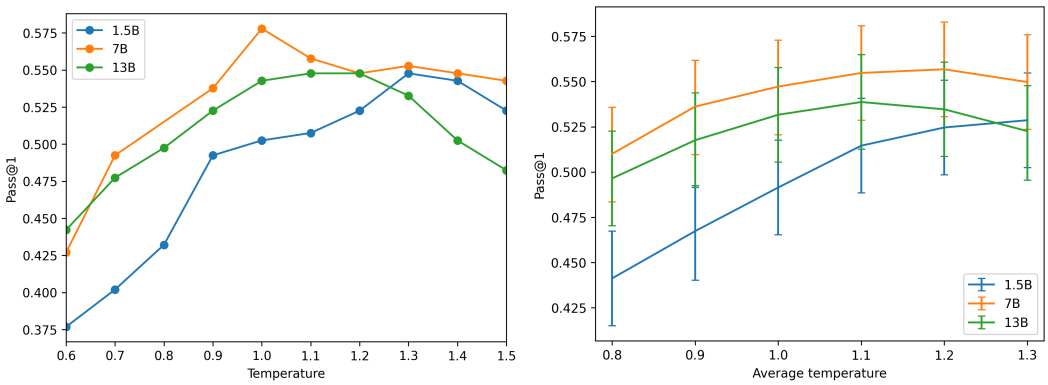


Figure 4: **Effect of sampling temperature.** We report Pass@1 scores of the 1.5B, 7B and 13B parameter model on 200 random samples from the validation set of the LeanDojo "random" benchmark by sampling temperature, using nucleus sampling with a probability mass `top_p` of 0.95. All runs use 64 tactic samples per search node. **Left:** Optimal temperatures for proof search are higher than what would be expected from the optimal temperatures for pass@k scores for code generation with models of comparable sizes [Chen et al., 2021]. **Right:** The same graph smoothened by averaging over five consecutive temperatures with 90 % confidence intervals, assuming a binomial distribution for the number of proved theorems among the 5×200 proof attempts.

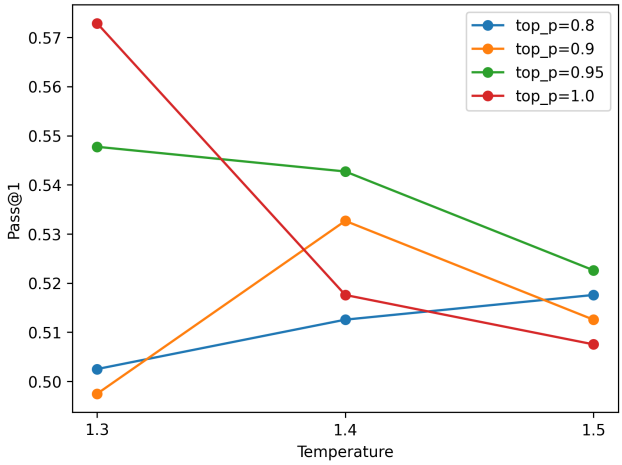


Figure 5: **Effect of nucleus sampling parameters.** We report Pass@1 scores of the 1.5B parameter model on 200 random samples from the validation set of the LeanDojo "random" benchmark by sampling temperature and nucleus sampling probability mass `top_p`. Smaller values for `top_p` do not allow to increase the sampling temperature beyond its optimal value of $T = 1.3$.

Figure 4 shows the effect of sampling temperature on proof success using 64 sampled tactics per node. Higher temperatures perform better due to the large number of sampled tactics and the importance of diversity in search-based benchmarks, with the proof rate reaching its maximum at $T = 1.3$ before decreasing at even higher temperatures. In Figure 5, we investigate whether temperatures $T > 1.3$ can be made useful by using nucleus sampling [Holtzman et al., 2020] with a smaller probability mass, which does not appear to be the case. In the end, we decided on the following temperatures for the evaluation of our models on the full LeanDojo "random" benchmark: temperature 1.3 for the

1.5B and 7B model, and 1.1 for the 13B model on the 600 s and 1200 s benchmarks, respectively. For all models, we keep the nucleus sampling probability mass at the frequently used default of 0.95.

7.2 Search breadth

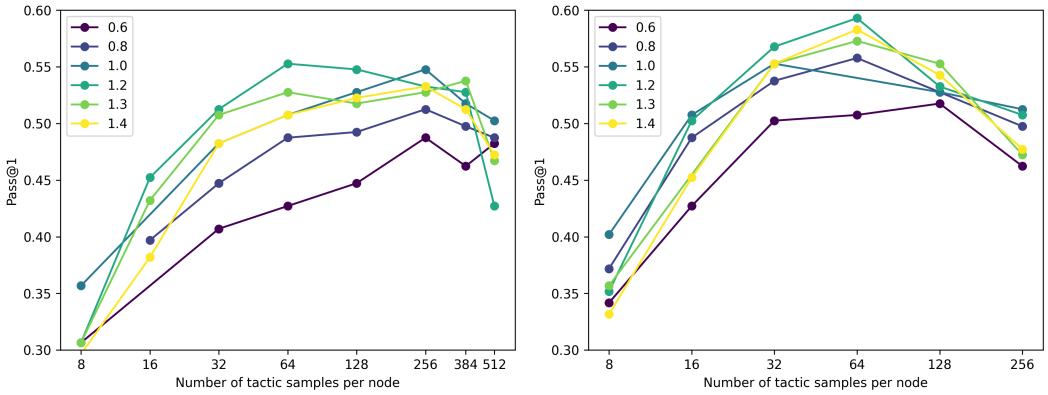


Figure 6: **Breadth vs. depth in tree-based proof search.** Proof rate by number of tactic samples per node. We report Pass@1 scores of the 1.5B and 7B parameter models, respectively, on 200 random samples from the validation set of the LeanDojo "random" benchmark by the number of tactic samples generated at each node in the search tree. A timeout of 600 seconds is imposed on each proof, resulting in a breadth vs depth tradeoff for the proof search. We report numbers for six different sampling temperatures in the range from 0.6 to 1.4 to account for the changing optima in the exploration-exploitation tradeoff characterized by each number of samples. Small models even benefit from generating very large numbers of samples while larger models have a more clearly defined optimum at 64 samples. This confirms the settings suggested by Yang et al. [2023]. Note that previous works use 32 [Polu and Sutskever, 2020] or up to 48 tactic samples per search node [Lample et al., 2022] on models in the 700M parameter range.

8 Training ablations

8.1 Pretraining distribution

Table 3: **Effect of the pretraining distribution.** We compare ppl@1.0 of the pretrained models as well as optimal values obtained in a preliminary hyperparameter grid search for finetuning Llama 2 7B [Touvron et al., 2023b] and Code Llama 7B [Rozière et al., 2023] on the proofstep prediction dataset described in Section 2.

pretraining	pretrained ppl@1.0	best ppl@1
Llama 2	3.569	1.571
Code Llama	2.672	1.496

8.2 Regularization methods

Table 4: **Effect of standard regularization techniques.** We report optimal validation perplexity ppl@T at various temperatures T. Default is with a weight decay coefficient of 0.1 and a proportion of pretraining rehearsal data of 0.0. Note in particular that dropout, which has been reported to be among the most effective regularization methods for language models [Xue et al., 2023], appears to be ineffective at lower temperatures but is effective at temperature-scaled perplexity ppl@1.6, which we have identified as most predictive for downstream prioving performance (Figure 8). We noticed this too late for model development, and leave the investigation of the interplay between dropout and temperature scaling for future work.

Method	Value	ppl@1.0	ppl@1.2	ppl@1.4	ppl@1.6	ppl@1.8
(default)		**1.318**	**1.298**	**1.299**	1.316	1.361
weight decay	0.00	1.323	1.302	1.300	1.316	1.353
weight decay	0.05	1.325	1.304	1.301	1.317	1.357
weight decay	0.20	1.321	1.300	1.301	1.322	1.370
weight decay	0.30	1.319	1.301	1.303	1.326	1.384
rehearsal	0.10	1.318	1.304	1.305	1.337	1.405
rehearsal	0.20	1.318	1.303	1.307	1.339	1.408
dropout	0.20	1.335	1.309	1.303	**1.308**	**1.327**

9 Additional experiments on temperature scaling

9.1 Proof success and temperature-scaled perplexity over the course of training

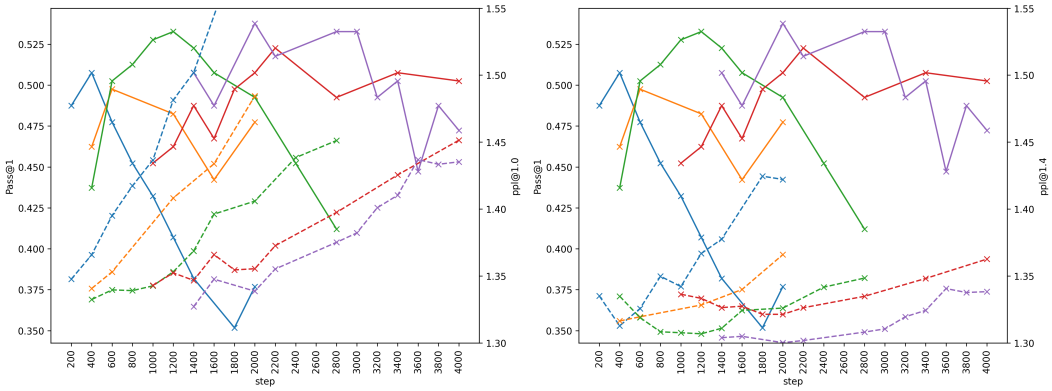


Figure 7: **Proof success rate (pass@1) and ppl@T over the course of training.** Solid lines depict pass@1 scores on 200 random samples from the validation set of the LeanDojo "random" benchmark, dashed lines perplexity evaluated at temperature $T = 1.0$ (left) and $T = 1.4$ (right). We evaluate a total of 45 checkpoints from five independent runs with varying learning rates and batch sizes, starting at the checkpoint that minimizes ppl@1.0. We do not evaluate all subsequent model checkpoints due to its high computational cost.

9.2 Predictiveness of temperature-scaled perplexity for proof success by temperature

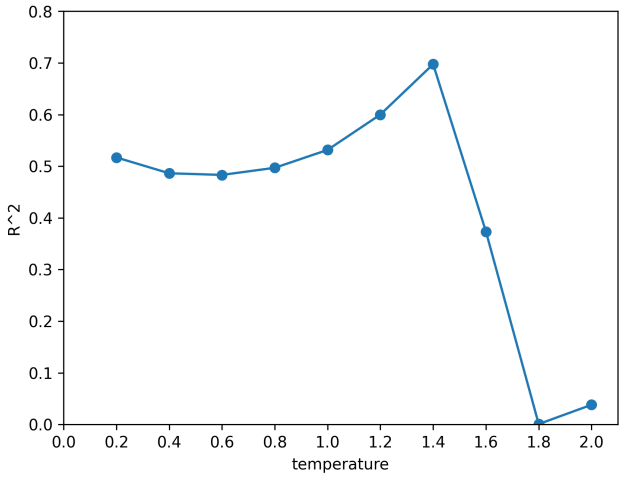


Figure 8: **Explained coviariance R^2 of ordinary least squares regressions for pass@1 by ppl@T for different values of T.** For examples and details on the regressed data, see Figure 1.

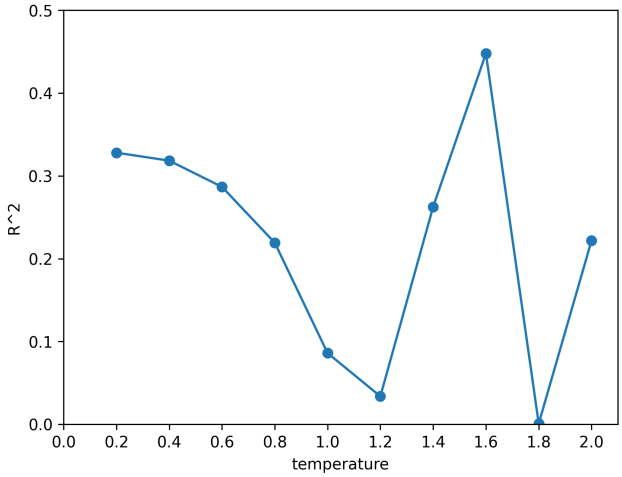


Figure 9: **Explained coviariance R^2 of ordinary least squares regressions for pass@1 by ppl@T for different values of T.** The regression only includes the 22 out of 45 points with the lowest values for ppl@T. These are the checkpoints considered for model selection in real-world applications. For examples and details on the regressed data, see Figure 1.

9.3 Minimal perplexity by temperature

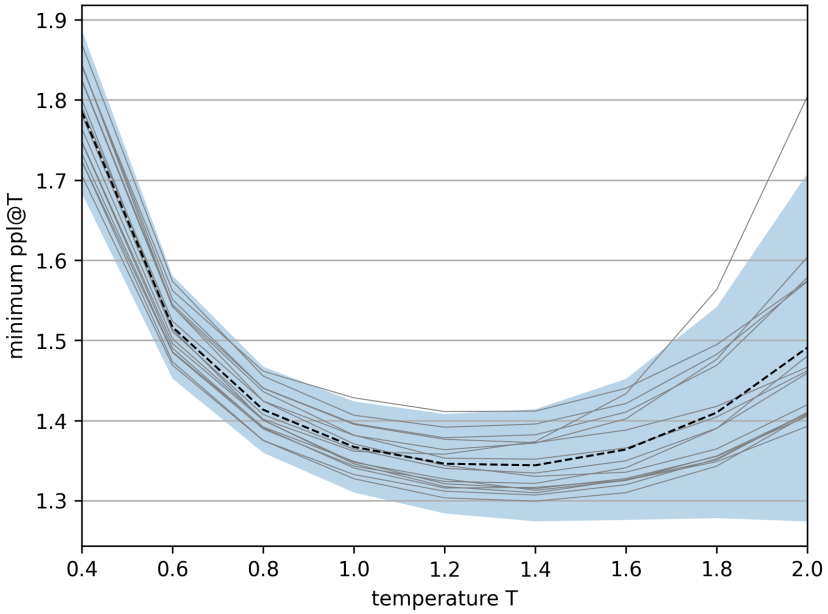


Figure 10: **Minimal ppl@T reached during training in dependency of T.** We show the average across 16 independent runs of a 1.5B model trained with varying learning rates and batch sizes and a 95 % two-sided confidence interval.

9.4 Temperature-scaled perplexity over the course of training

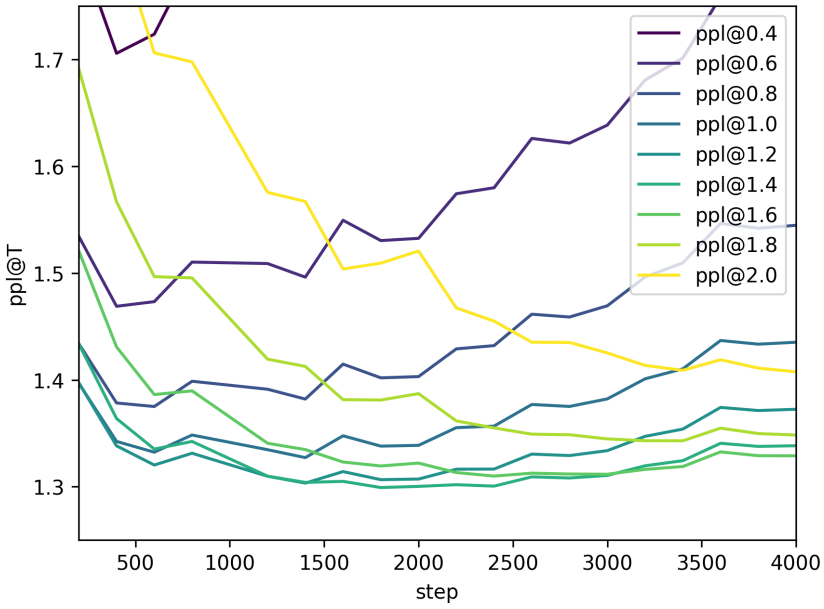


Figure 11: **Temperature-scaled perplexity ppl@T over the course of training.** Evaluated on a 1.5B parameter model.

10 Discussion of temperature scaling

In this section, we discuss theoretical reasons for the empirical usefulness of temperature scaling reported in Section 3. Whereas optimizing for ppl@T for temperatures $T > 1$ selects later checkpoints than ppl@1.0 (Figure 11) and could be viewed as deliberate overfitting with respect to standard perplexity, we would like to argue here that temperature scaling pursues a well-defined objective by offering the following complementary theoretical explanations.

Tree-search favors diversity. While validation perplexity evaluates tactic suggestion models on single given tactic prediction cases, proof search evaluates akin to pass@k metrics for code models, where the optimal temperature increases with k [Chen et al., 2021]. In our experiments, we found that a large temperature of 1.3 works best for models of size 1.5B and 7B parameters, and consequentially, model selection should take this into account.

Temperature scaling regularizes multi-epoch training. Multi-epoch training is beneficial on small datasets as it allows for better convergence, but it comes at a cost: the training data observed by the model is no longer an independent sample of a background distribution underlying both the training and validation dataset. Rather, oversampling models a sharper distribution than the original one, leading to a mismatch between the modelled distribution and the one required for downstream applications. Increasing the temperature can therefore be seen as a simple method to undo this sharpening of the modelled distribution, bringing it back closer to the target distribution. In Section 10.1, we express this intuition as optimizing the trade-off between optimization error incurred from not taking enough gradient steps and sampling error from the distribution mismatch: temperature scaling recalibrates the model after long multi-epoch training, allowing smaller optimization error while diminishing the increase of sampling error. In Figure 12, we provide a sketch of an analogy for this behavior.

10.1 Temperature scaling and the training error / sampling error trade-off

Let $(X, Y) \sim \mathcal{D}$ be the joint distribution belonging to the prediction task $Y|X$, where $Y \in \Delta_{N-1}$ belongs to the space of categorical distributions over N classes. Let $P(X, Y)$ be the prior of a pretrained model for this distribution and $D \sim \mathcal{D}^m$ a collection of m indepdendent samples from $\mathcal{D}$. Let nD denote the n-fold disjoint union of D, i.e. the dataset consisting of n *epochs* of D. Given a dataset D, let P_D denote the Bayesian posterior distribution from prior P and observations D and by P_D^t the result of training P with a cross-entropy loss on D in order to approximate P_D. Denoting cross-entropy loss as

$$H(P, Q) = \mathbb{E}_{(X,Y)\sim P}[-\log Q(Y|X)],$$

we have a dataset-size dependent monotonic convergence of the *optimization error* $H(\mathcal{D}, P_D^t) - H(\mathcal{D}, P_D)$, the gap between $H(\mathcal{D}, P_D^t)$ and the Bayes optimal loss $H(\mathcal{D}, P_D)$:

$$H(\mathcal{D}, P_D^t) - H(\mathcal{D}, P_D) \to 0 \qquad \text{with } |D| \to \infty.$$

Multi-epoch training means passing from D to nD for some $n \in \mathbb{N}$, and we can decompose

$$H(\mathcal{D}, P_{nD}^t) = \big(H(\mathcal{D}, P_{nD}^t) - H(\mathcal{D}, P_{nD})\big) + H(\mathcal{D}, P_{nD}),$$

where the first term

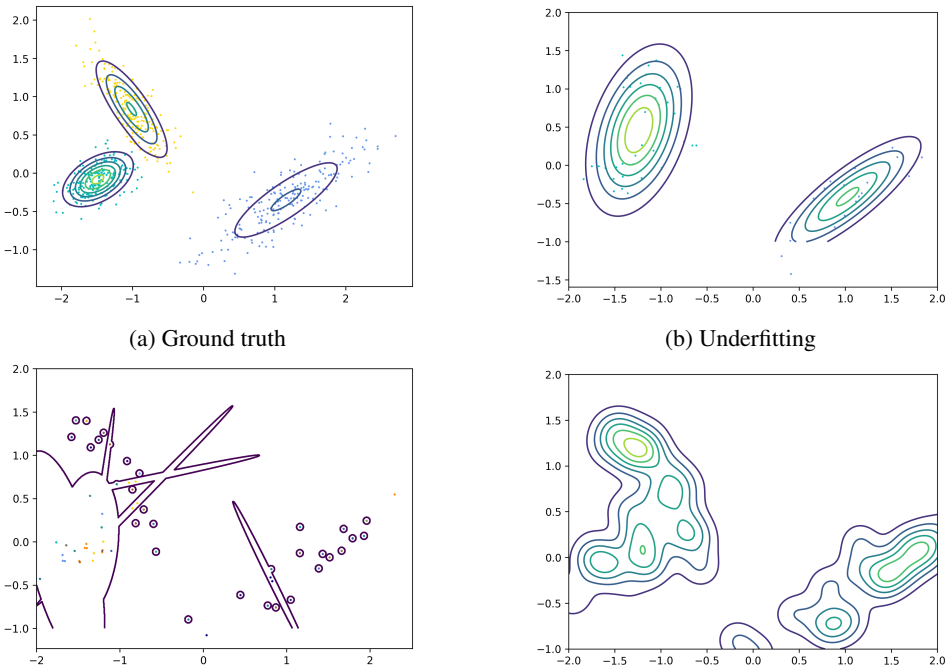
$$H(\mathcal{D}, P_{nD}^t) - H(\mathcal{D}, P_{nD})$$

is the optimization error discussed above, and $H(\mathcal{D}, P_{nD})$ denotes the *sampling error* arising from pretending that nD was an independent sample of $\mathcal{D}$.

In the limit $n \to \infty$, the joint distribution $P_{nD}(X, Y)$ concentrates to the empirical distribution of D and $P_{nD}(Y|X)$ is undefined outside this support. This *need not* impair generalization performance of $P_{nD}^t(Y|X)$ but very likely does so as neural networks do not come with error bounds on out of domain generalization, and are known to allocate their computational resources in line with the input data densities $P_{nD}(X)$. Assuming that it does, however, we have, say,

$$H(\mathcal{D}, P_{nD}) \to \infty \qquad \text{with } n \to \infty$$

monotonically, and the loss $H(\mathcal{D}, P_{nD}^t)$ decomposes into an optimization error term that monotonically decreases with n and a sampling error term that monotonically increases with n, which we call the optimization error / sampling error trade-off.



(a) Ground truth

(b) Underfitting

(c) Overfitting

(d) *Temperature-scaled* overfitting

Figure 12: **Sketch of an analogy to explain temperature scaling.** 12a A mixture of Gaussians with three components is to be modelled from a small amount of samples. 12b If not trained for enough epochs, a machine learning model may underfit the distribution. This is depicted here by fitting a Gaussian mixture model with two components. 12c A machine learning model trained on the small amount of samples for too many epochs may overfit the distribution, converging to the empirical distribution of the dataset in the limit. This is depicted here by repeating the samples several times and fitting a Gaussian mixture model with a large number of components. 12d From such an overfitted model, a better model can be obtained by recalibrating the confidence. This is depicted here by multiplying the covariance matrices of the components of the Gaussian mixture model by a large constant. We believe that temperature scaling has a similar effect in the output space of categorical distributions as covariance scaling has in the case of Euclidean output space as in this example.

We believe that temperature scaling operates on this trade-off by lowering the increase of sampling error in dependency of n, thereby moving the minimum to larger n and lower overall loss.

10.2 Comparison of temperature scaling to other distribution adaptation techniques

Comparison to label smoothing

Label smoothing is a generalization technique that aims to prevent overfitting by replacing a one-hot target distribution with its convex combination with a uniform distribution over the labels. Whilst superficially similar to temperature scaling, models trained with label smoothing will still converge to the empirical distribution in the multi-epoch limit without guarantees on smoothness outside the support of the training dataset. Temperature scaling, on the other hand, explicitly aims at smoothness outside of the support and implicitly smoothens predictions on the support as a byproduct. In other words, label smoothing is recommended to account for noise in the training dataset labels [Goodfellow et al., 2016], not for generalization error induced by multi-epoch training.

Comparison to softmax tempering

Softmax tempering, originally proposed in the context of knowledge distillation [Hinton et al., 2014], is the practice of applying a temperature to the pre-softmax model logits during training. Dabre and Fujita [2020] suggest to use a temperature $T > 1$ for training neural machine translation models

on low-resource language pairs with a dataset of 18,088 sentences. The authors argue that this reduces overfitting by smoothening the distribution, and observe that such tempered models behave similarly in beam search as in greedy decoding. We would like to explain these findings as follows: if in an untempered model, a specific model logit on a specific logit z_t converges to $z_t \to z$ with training time $t \to \infty$, then, in a tempered model with temperature T, we will likewise have $\frac{z_t}{T} \to z$ with $t \to \infty$. In other words, softmax tempering is not an architecture change but a change in the initialization of the output layer. Moreover, artificially smoothening the training distribution during training before computing the loss has the effect of *sharpening* the distribution during inference, i.e. it is architecture-wise equivalent to use a temperature of $\frac{1}{T}$ in a untempered model as to use a temperature of 1 in one tempered with temperature T. This explains the observed similarity of sampling and greedy decoding. As to the training dynamics and the question of overfitting, the change of initialization of course does have an effect: it increases the number of steps until the model has memorized the training set with predicted probabilities of almost one and vanishing gradients, and produces an even sharper distribution on the training data at inference time. In other words, we believe that softmax tempering with temperatures $T > 1$ has the effect of *increasing* overfitting, and that this can be useful up to a certain point in order to trade off optimization error and sampling error (Appendix 10.1).

Li et al. [2022b], conversely, suggest to use tempering with a temperature of $T = 0.2 < 1$ during finetuning. This has the effect of sharpening the training distribution and smoothening the inference distribution. In order to counter this effect, the authors decide to use a temperature of 0.12 at inference time, resulting at an effective temperature of $\frac{0.12}{0.2} = 0.6$ at inference which is in line with previously observed values for optimal temperature for large scale code sampling [Chen et al., 2021]. We are not aware of a rationale for tempering with temperatures $T < 1$ in data constrained finetunings as this appears to shorten the training time in a regime where the optimization error is likely to dominate.

Our proposed method of temperature scaling leaves the training distribution unchanged and instead adapts the criterion for early stopping. However, putting the different initialization aside, early stopping based on ppl@T, $T > 1$ is equivalent to usual early stopping based on ppl@1.0 on a model tempered with temperature T.

11 Proof search metrics by model size

We evaluate our models on the following metrics.

End-to-end proving metrics. Proof success rate (`pass@1`), number of nodes in the proof graph (`num_total_nodes`), number of expanded nodes (`num_searched_nodes`), total tactic execution time per proof (`env_time`, in seconds), total tactic generation time per proof (`actor_time`, in seconds) and total time per proof (`total_time`, in seconds). These metrics evaluate a proving system end-to-end and provide insights into its resource consumption and breadth of search.

Tactic quality metrics. Tactic success rate of deduplicated tactics (`tactic_success`), proof length in lines (`proof_length`) and characters (`proof_chars`). These metrics allow to evaluate the accuracy of single tactic predictions.

Tactic diversity metrics. Tactic diversity measured as the fraction of the number of unique tactics per node and the number of generations per node (`tactic_diversity`, head symbol diversity of the first words of the deduplicated tactics (`head_tactic_diversity`), word diversity of the deduplicated tactics as measured by average Jaccard similarity between two tactics from the same node split into words at punctuation characters [3] (`tactic_jaccard`). These metrics provide a measure of the diversity of the generated tactics.

Table 5 reports the metrics we obtained on our 1.5B, 7B and 13B parameter models.

Table 5: **Proving metrics by model size.** Using the tuned temperatures of 1.3, 1.3 and 1.1 for the 1.5B, 7B and 13B model respectively. With increasing model size, the tactic generation models produce higher quality tactic suggestions and rely less on search. The 7B model strikes the overall best balance between generation speed and quality. We report means and include the median where it differs considerably from the mean.

Metric	1.5B	7B	13B
pass@1	56.2	57.7	51.5
proof_length_avg	2.006	1.835	1.624
proof_chars_avg	46.437	46.748	41.211
num_total_nodes_avg	1119.761	616.160	141.536
num_total_nodes_median	857	544	97
num_searched_nodes_avg	30.222	17.458	4.215
num_searched_nodes_median	27	17	4
env_time_avg	72.355	41.371	13.013
env_time_median	45.104	26.592	7.010
actor_time_avg	276.960	357.806	410.854
actor_time_median	267.681	383.814	340.943
total_time_avg	340.982	377.089	338.284
total_time_median	365.711	419.838	262.547
tactic_success_avg	0.147	0.149	0.196
tactic_diversity_avg	0.782	0.802	0.693
tactic_diversity_median	0.844	0.875	0.766
head_tactic_diversity_avg	0.256	0.237	0.190
head_tactic_diversity_median	0.231	0.208	0.159
tactic_jaccard_avg	0.115	0.119	0.159

[3] We use the following characters for splitting: ␣ . , : ; [] () {} ⟨⟩ ↦ $, where ␣ denotes a space character.

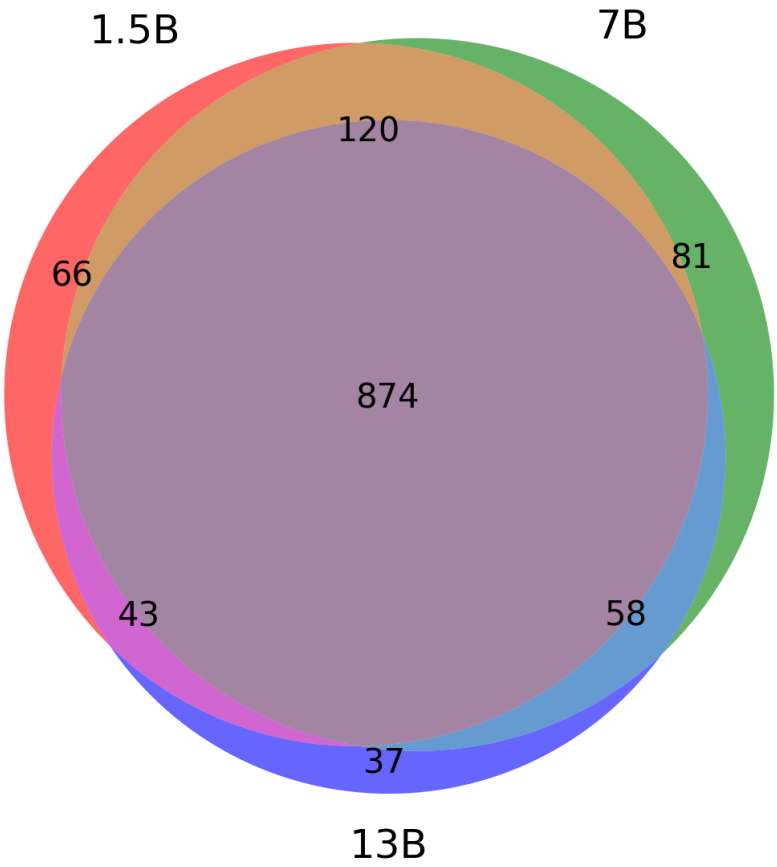


Figure 13: **Comparison of solved theorems by differently sized models.** Evaluated on the full LeanDojo "random" benchmark under a 600 second wall clock time limit per theorem.