Towards Scaling Laws for Symbolic Regression

David Otte[1] **Jörg K.H. Franke**[1,2,3,4] **Frank Hutter**[1,2,5]

[1]University of Freiburg [2]ELLIS Institute Tübingen
[3]Open-Sci Collective [4]LAION [5]Prior Labs

Abstract

Symbolic regression (SR) aims to discover the underlying mathematical expressions that explain observed data. This holds promise for both gaining scientific insight and for producing inherently interpretable and generalizable models for tabular data. In this work we focus on the basics of SR. Deep learning-based SR has recently become competitive with genetic programming approaches, but the role of scale has remained largely unexplored. Inspired by scaling laws in language modeling, we present the first systematic investigation of scaling in SR, using a scalable end-to-end transformer pipeline and carefully generated training data. Across five different model sizes and spanning three orders of magnitude in compute, we find that both validation loss and solved rate follow clear power-law trends with compute. We further identify compute-optimal hyperparameter scaling: optimal batch size and learning rate grow with model size, and a token-to-parameter ratio of $\approx$15 is optimal in our regime, with a slight upward trend as compute increases. These results demonstrate that SR performance is largely predictable from compute and offer important insights for training the next generation of SR models.

1 Introduction

Symbolic regression seeks to uncover the underlying mathematical expressions that describe the relationship between a set of observed variables. In recent years, pre-trained transformer models for symbolic regression have become increasingly popular and have started to achieve performance comparable to classic genetic programming methods [1–4]. We observe that prior work has focused on tweaking the training process while holding the scale mostly constant. To date, we are not aware of any symbolic regression models trained with more than $\approx$100 million parameters.

Motivated by the impact of scaling laws in language modeling [5, 6], we ask: *Do similar scaling laws exist for symbolic regression, and can they have a comparable impact on future model design?* To answer this question, we see the need for a more controlled and scalable training setup. Specifically, we propose (1) a more systematic synthetic data generation approach that allows for stricter control over the expressions seen during training, and (2) targeted architectural improvements motivated by recent tabular foundation models. To keep the scope manageable yet insightful, our expressions contain only integer constants and at most two variables, while still covering a broad operator mix commonly seen in symbolic regression. Based on this setup, we conduct the first systematic scaling study in symbolic regression, spanning five model sizes and three orders of magnitude in compute. Our goal is not to beat existing baselines, but to establish that symbolic regression with transformers follows predictable scaling laws, which can serve as a design principle for future models.

Our key contributions are: (1) We demonstrate power-law scaling of solved rate and loss with compute over three orders of magnitude. (2) We identify systematic trends in optimal learning rate, batch size, and token-to-parameter ratio. (3) We introduce a scalable end-to-end pipeline that allows a clean and efficient scaling analysis.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

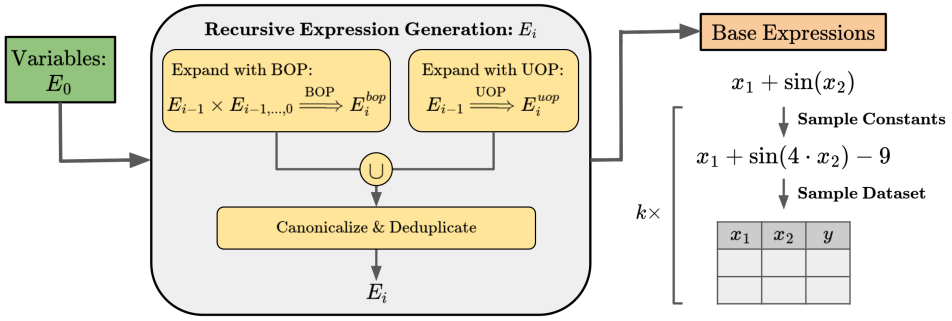


Figure 1: Overview of our two-step data generation. In the first step, we recursively generate a set of base expressions by applying a set of binary operators (BOP) and a set of unary operators (UOP). In the second step, we sample expression-dataset pairs from our base expressions.

2 Related Work

Biggio et al. [1] were the first to suggest pre-training transformers on millions of synthetic symbolic regression tasks. They use a set-encoder and a standard decoder to autoregressively predict function skeletons, whose constants are then refined via BFGS. Subsequent work has explored improvements on this theme, for example sharpening skeleton separation with a contrastive term [7] or framing the problem as multimodal translation [3]. Other approaches, such as E2E [2] and SymFormer [8] propose end-to-end architectures to directly output full expressions including constants, simplifying the pipeline and achieving comparable performance with genetic programming approaches. Our study builds on this end-to-end paradigm, coming closest to E2E, but shifts the focus from loss engineering to the role of scale.

3 Methodology

In this section, we describe our approach for data generation and model training. We adopt many ideas from Kamienny et al. [2], but apply targeted improvements that allow for more efficient training and a better scaling analysis.

Data generation Fig. 1 shows our two-step approach for data generation. While previous work usually adopts the expression sampling mechanism proposed by Lample and Charton [9], we generate a complete set of expressions recursively, starting with just the variables and then iteratively applying unary and binary operators. This allows us to get more control over the expressions the model sees during training, especially in terms of the number of duplicates, and avoids biasing the model towards certain, more common expressions. Furthermore, we filter out equivalent expressions and unify their representation, allowing us to feed the model with cleaner training data. We obtain our set of base expressions E by taking all expressions up to a fixed threshold. For each of these expressions, we try to sample k expression-dataset pairs. Therefore, we first insert random constants and then sample a random dataset from a Gaussian mixture following Kamienny et al. [2]. More details about our data generation can be found in App. A.

Tokenization Similar to existing approaches, we decide to use base 10 floating-point notation to encode the tabular dataset. Thus, we split each numeric value into a mantissa, including the sign, and an exponent. Under the assumption that different representations of expressions are equally difficult to learn for the model, we represent our target expressions as LaTeX strings for better readability. In the target expression, we tokenize all constants digit-wise.

Architecture Given the great scaling properties of transformers [10, 5], we mainly adopt the standard encoder-decoder architecture used by previous work [1, 2] and only apply targeted improvements. Fig. 2 shows an overview of our architecture. While previous work decided to merge each input point into a single embedding, we choose a different embedding strategy and aim to get a single embedding for each cell in our dataset. Therefore, we up-project our mantissas and exponents to the embedding dimension and simply sum them together. This embedding strategy allows us to take inspiration from recent tabular foundation models and adopt their encoder architecture, which has

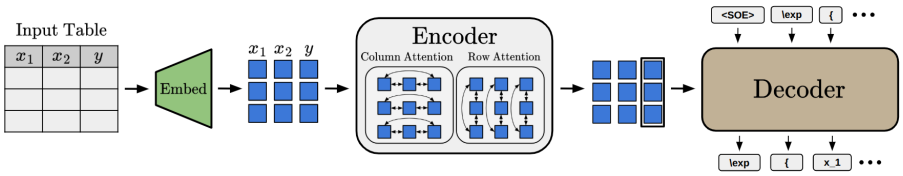


Figure 2: Overview of our model architecture.

proven to be very powerful for handling tabular data [11, 12]. The main innovation here is that in each layer, we now apply both row- and column-wise attention across different variables and different data points. Our decoder follows the standard sequence-to-sequence transformer architecture [13] and only cross-attends to the updated embeddings of the target cells. We train with cross-entropy loss between the predicted tokens and the tokens of the true expression.

Experiments To analyze scaling, we follow the strategies of scaling law papers in language modeling [5, 6, 14]. To approximate training compute, we adapt the FLOP estimate of Kaplan et al. [5] to encoder-decoder models: FLOPs $\approx 6 \cdot (N_{enc} \cdot D_{in} + N_{dec} \cdot D_{out})$, with $N = N_{enc} + N_{dec}$ being the number of feed-forward parameters and D_{in} and D_{out} denoting the number of input and output tokens seen during training. For each of multiple model sizes, we conduct a sweep over different batch sizes and learning rates using a token-to-parameter ratio of 20, which was found to be optimal for language model training [6]. We then retrain the models with the lowest validation loss per model size with varying token-to-parameter ratios.

4 Experimental Results

4.1 Setup

We use models of five different sizes (6.5M-93M parameters) and token-to-parameter ratios ranging from 5 to 80. For our training data, we generate $|E| = 100{,}000$ expressions with up to two variables. For each expression, we then sample up to $k = 3{,}600$ expression-dataset pairs by inserting random integer constants into the expression and sampling datasets of 64 points. We also generate independent validation and test splits, each containing $1{,}000$ expressions sampled from E but with freshly sampled constants and datasets. This ensures that no expression-dataset pair from training appears in evaluation.

We found that the AdamCPR optimizer [15], combined with a linear warm-up of the learning rate for the first 5% of steps and cosine annealing thereafter, works best for our setting. All training hyperparameters are reported in App. C. For evaluation, we first randomly sample 128 expressions from the model and keep the one with the highest R^2 (coefficient of determination) score. Then, we look at both the perfect-solved ratio, denoted as $\text{Acc}_{\text{solved}}$, and the expression ratio with $R^2 > 0.99$, denoted as $\text{Acc}_{R^2>0.99}$, on the test split. We run each evaluation over three different seeds and report the mean $\text{Acc}_{\text{solved}}$ and $\text{Acc}_{R^2>0.99}$ across these seeds.

4.2 Results

We report the results of our hyperparameter sweep in App. B (Fig. 4). Detailed evaluation results of the trainings with the best found hyperparameters can be found in App. B (Fig. 2). In the following, we highlight our three key findings.

Solved rate and loss scale as power laws with compute To analyze the effect of scaling on performance, we look at training with the optimal configuration per model size. Following the approach of Hoffmann et al. [6], we get a Pareto front of our models by binning compute into $1{,}500$ intervals and keeping the models with the lowest validation loss up to their compute level. We adopt the assumption that scaling trends follow power-law trends and thus fit power laws on the remaining data. Fig. 3 shows the results for the $\text{Acc}_{\text{solved}}$ metric, which follows power-law scaling with compute across three orders of magnitude. In App. B, we report analogous plots for $\text{Acc}_{R^2>0.99}$ and test loss, yielding similar insights. Larger models with more data consistently improve the solved ratio and reduce loss, with no signs of saturation in the largest models tested. For instance, we can see that $\text{Acc}_{\text{solved}}$ increases from approximately 0.03 at the lowest compute budget to 0.6 at the highest

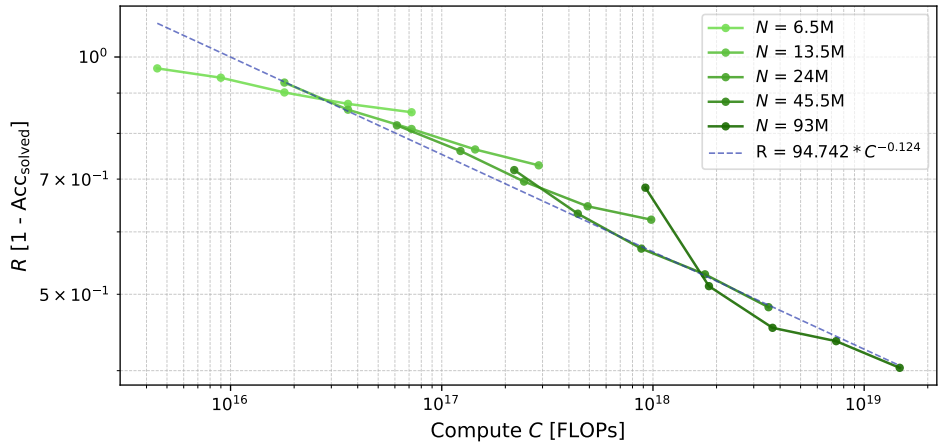


Figure 3: $\text{Acc}_{\text{solved}}$ scales as a power law of training compute. Each marker corresponds to a trained model and depicts the mean perfect-solved ratio over three random seeds. Plot design inspired by Franke et al. [16].

compute budget, and that according to the scaling law, we could reach 0.8 with a compute budget of 3.8×10^{21} FLOPs. Also, by comparing the scaling laws, we can see that improvements in $\text{Acc}_{R^2>0.99}$ happen much faster than in $\text{Acc}_{\text{solved}}$, highlighting how hard it is to achieve exact expression matches.

Optimal batch size and learning rate grow with compute Based on the results of our grid search, we analyze the scaling trends of the optimal learning rate and batch size. We follow the strategy of Porian et al. [14], which involves two-step interpolation by first interpolating the optimal learning rates and then the optimal batch sizes using Akima interpolation, followed by fitting scaling laws to both the optimal interpolated values. App. B (Fig. 5) shows systematic trends for the compute-optimal choices of batch size and learning rate, indicating both should increase when scaling compute. The upward trend for the learning rate contrasts with findings for LLMs, showcasing how training characteristics can vary between different tasks. While we can see clear tendencies, we also note that more training runs over multiple seeds are needed to further reduce variance.

Optimal trade-off between model size and data size We fit scaling laws to the optimal number of parameters and expression tokens per compute budget. The results are shown in App. B (Fig. 6). We look at the ratio of the predicted scaling laws, noticing an increasing trend with growing compute. Although a token-to-parameter ratio of $\approx$15 seems to be optimal in our compute regime, the results indicate that the training dataset size should scale slightly faster than the model size.

5 Limitations and Conclusion

Our findings come with several limitations. We restrict the domain to expressions with at most two variables and small integer constants, whereas real-world symbolic regression tasks often involve more variables and floating-point constants. Thus, our predicted scaling laws might not directly translate to other symbolic regression setups. Due to computational constraints, we perform only single-seed training runs, which may introduce variance in our results. Moreover, our analysis spans a limited compute range, so extrapolations beyond this remain unverified. Finally, we do not compare against other symbolic regression methods, as we focus on general scaling insights, and comparison is difficult due to our customized setup.

Despite our limitations, our results show that symbolic regression with transformers follows predictable scaling laws. Performance improves with compute according to power laws, the optimal token-to-parameter ratio is approximately 15 for our compute budgets, and both batch size and learning rate should increase with model size. These insights provide practical heuristics for future work and suggest a new way of significantly improving model performance. Future work should extend our analysis to more complex expressions and verify that end-to-end symbolic regression with improved data generation, model architecture, and scaling can outperform all other approaches. We hope this work encourages a more systematic approach to scaling in symbolic regression.

Acknowledgements

The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG. This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828. We acknowledge funding by the European Union (via ERC Consolidator Grant DeepLearning 2.0, grant no. 101045765). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pages 936–945. Pmlr, 2021.

[2] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. *Advances in Neural Information Processing Systems*, 35:10269–10281, 2022.

[3] Kazem Meidani, Parshin Shojaee, Chandan K Reddy, and Amir Barati Farimani. Snip: Bridging mathematical symbolic and numeric realms with unified pre-training. *arXiv preprint arXiv:2310.02227*, 2023.

[4] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*, 2021.

[5] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[6] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[7] Wenqiang Li, Weijun Li, Linjun Sun, Min Wu, Lina Yu, Jingyi Liu, Yanjie Li, and Songsong Tian. Transformer-based model for symbolic regression via joint supervised learning. In *The Eleventh International Conference on Learning Representations*, 2022.

[8] Martin Vastl, Jonáš Kulhánek, Jiří Kubalík, Erik Derner, and Robert Babuška. Symformer: End-to-end symbolic regression using transformer-based architecture. *IEEE Access*, 12:37840–37849, 2024.

[9] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*, 2019.

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[11] Lars Lorch, Scott Sussex, Jonas Rothfuss, Andreas Krause, and Bernhard Schölkopf. Amortized inference for causal structure learning. *Advances in Neural Information Processing Systems*, 35: 13104–13118, 2022.

[12] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[14] Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. *Advances in Neural Information Processing Systems*, 37:100535–100570, 2024.

[15] Jörg KH Franke, Michael Hefenbrock, Gregor Koehler, and Frank Hutter. Constrained parameter regularization. 2023.

[16] Jörg KH Franke, Urs Spiegelhalter, Marianna Nezhurina, Jenia Jitsev, Frank Hutter, and Michael Hefenbrock. Learning in compact spaces with approximately normalized transformers. *arXiv preprint arXiv:2505.22014*, 2025.

[17] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

[18] Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, et al. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023.

A Data generation details

A.1 Expression generation

Recursive generation Our general goal is to get a complete set of unique symbolic expressions up to a certain size. We try to approach this by looking at the depth of the corresponding expression trees. We start with all expressions of depth 0, which are just our variables: $E_0 = \{x_1, ..., x_n\}$. Given all expressions up to a certain depth, we can get all possible expressions of the subsequent depth by: (1) Applying all unary operators to all expressions in E_{i-1} and (2) applying all binary operators to every pair of expressions in E_{i-1} and $E_{i-1,...,0} = E_{i-1} \cup \cdots \cup E_0$.

Canonicalization and deduplication For having consistent training data and avoiding duplicates, we try to bring every generated expression into a canonical form using Sympy [17]. We avoid using their simplify function, as it is based on heuristics and gives inconsistent outputs. Instead, we expand all expressions and iteratively check for specific simplifications we can do. Finally, we apply a custom ordering function. For deduplication, we can now simply check if a new expression was already found before and add it to our set of new expressions.

A.2 Data sampling

We continue with the first base expressions of $E_0, ..., E_d$ up to a certain threshold. This means taking the full generated sets of lower depths and sampling expressions of the final depth we want to consider until we reach the threshold. Now, for each of the base expressions we try to sample k expression-dataset pairs. For each sampled pair, we first insert random constants into the base expression, by sampling a multiplicative and an additive constant for each variable and each unary operator by a probability p. Then, we sample a dataset following Kamienny et al. [2], with retrying up to five times in case of failure. Specifically, this approach involves: (1) Sampling the number of clusters and weights for each cluster. (2) Sampling centroids, variances and a distribution shape (Gaussian or uniform) for each cluster. (3) Sampling a dataset and applying a randomly sampled rotation from the Haar distribution.

Table 1: Used hyperparameters for data generation.

Parameter	Value
Number of variables	2
Max tree depth	3
BOP	{+, -, ·, ÷}
UOP	{exp, sin, neg, sqrt}
$\|E\|$	100,000
k	3,600
Data points per dataset	64
Constant type	integer
Constant range	-9 to 9
Constant probability p	0.2
Max number of clusters	5
Dataset sampling retries	5

B Additional Results

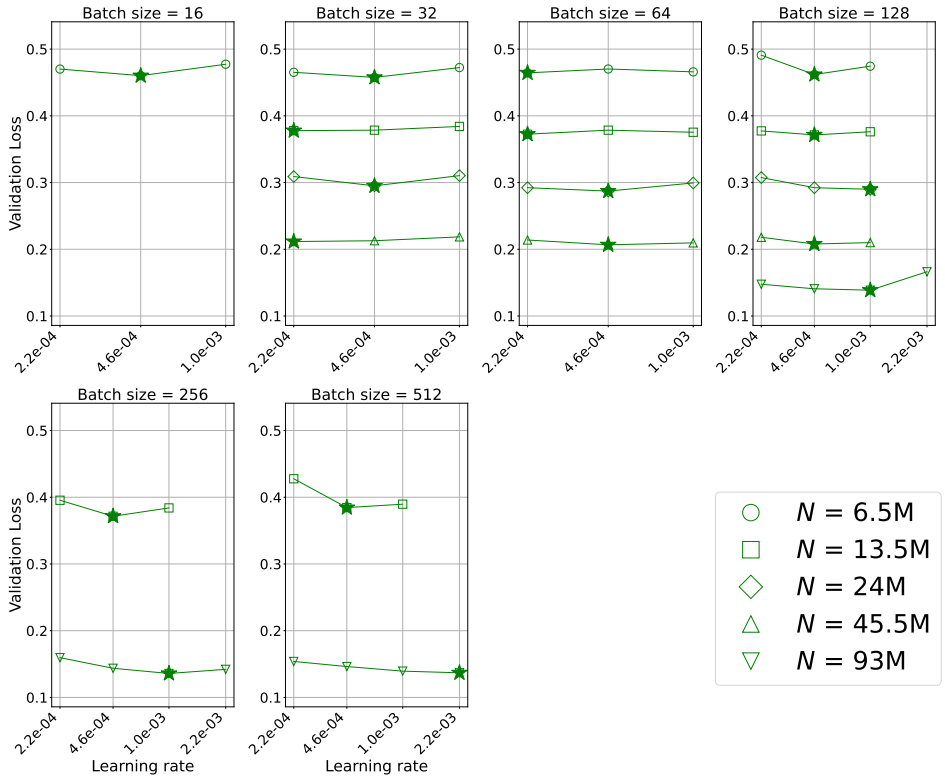


Figure 4: Results of our hyperparameter grid search. For each model size we trained different combinations of batch size and learning rate using a token-to-parameter ratio of 20, until we found an optimum. The stars indicate the runs with the lowest loss for each model size and batch size. Plot design inspired by Wortsman et al. [18].

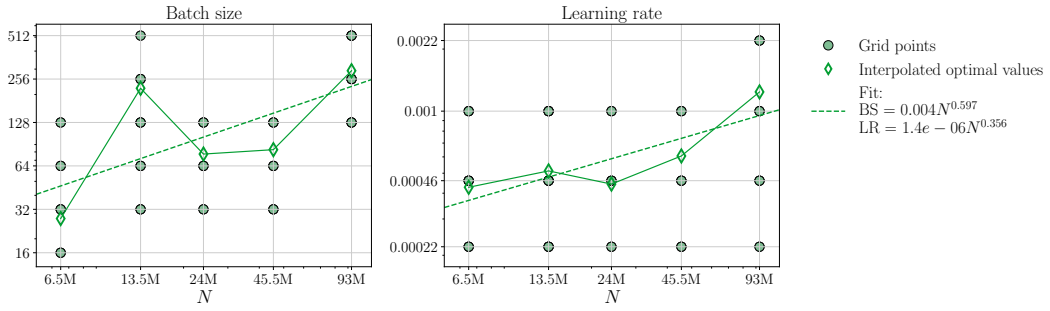
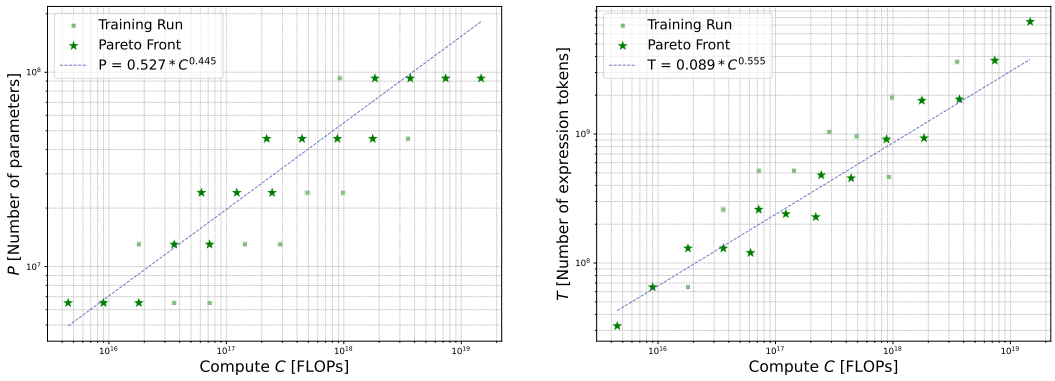


Figure 5: Compute-optimal batch size and learning rate as functions of model size N.



(a) Compute-optimal number of parameters

(b) Compute-optimal dataset size

Figure 6: Both optimal number of parameters and training tokens scale as power laws with compute.

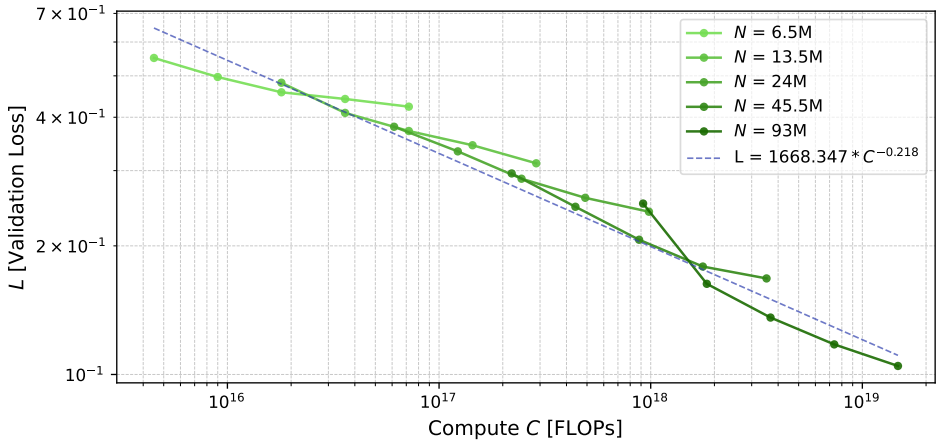


Figure 7: Validation loss scales as a power law of training compute. Each marker corresponds to a trained model and depicts the final validation loss after training.

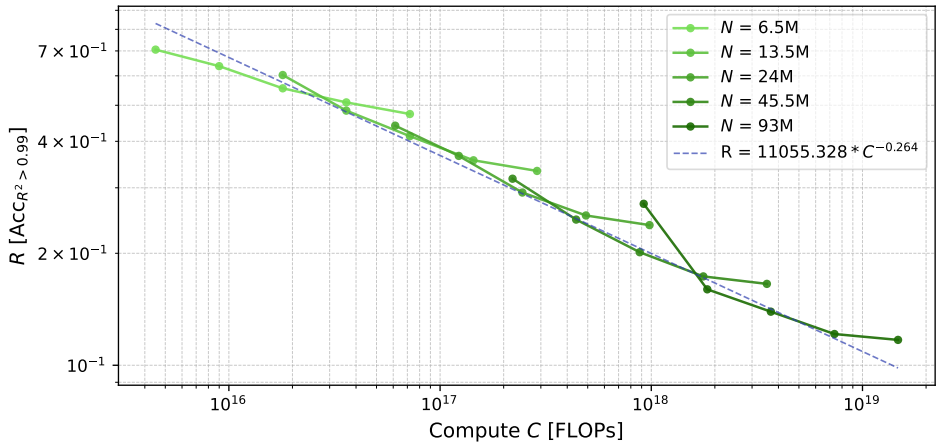


Figure 8: $\text{Acc}_{R^2>0.99}$ scales as a power law of training compute. Each marker corresponds to a trained model and depicts the mean $\text{Acc}_{R^2>0.99}$ over three random seeds.

Table 2: Full result table of runs with best found hyperparameters.

Model Size	Batch Size	Learning Rate	Training FLOPs	$\text{Acc}_{\text{solved}}$	$\text{Acc}_{R^2>0.99}$	Final Validation Loss
6.5M	32	4.6e-4	4.50e+15	0.0327	0.2943	0.5506
6.5M	32	4.6e-4	8.99e+15	0.0587	0.3630	0.4969
6.5M	32	4.6e-4	1.80e+16	0.0983	0.4447	0.4576
6.5M	32	4.6e-4	3.60e+16	0.1280	0.4910	0.4414
6.5M	32	4.6e-4	7.20e+16	0.1490	0.5263	0.4235
13.5M	128	4.6e-4	1.80e+16	0.0720	0.3970	0.4818
13.5M	128	4.6e-4	3.60e+16	0.1427	0.5163	0.4095
13.5M	128	4.6e-4	7.20e+16	0.1897	0.5870	0.3714
13.5M	128	4.6e-4	1.44e+17	0.2367	0.6443	0.3440
13.5M	128	4.6e-4	2.88e+17	0.2713	0.6670	0.3121
24M	64	4.6e-4	6.13e+16	0.1810	0.5593	0.3800
24M	64	4.6e-4	1.23e+17	0.2401	0.6345	0.3328
24M	64	4.6e-4	2.45e+17	0.3050	0.7087	0.2872
24M	64	4.6e-4	4.91e+17	0.3535	0.7473	0.2591
24M	64	4.6e-4	9.81e+17	0.3783	0.7619	0.2404
45.5M	64	4.6e-4	2.20e+17	0.2813	0.6827	0.2955
45.5M	64	4.6e-4	4.41e+17	0.3673	0.7536	0.2469
45.5M	64	4.6e-4	8.81e+17	0.4287	0.7987	0.2067
45.5M	64	4.6e-4	1.76e+18	0.4700	0.8267	0.1789
45.5M	64	4.6e-4	3.53e+18	0.5185	0.8345	0.1678
93M	256	1.0e-3	9.21e+17	0.3173	0.7283	0.2512
93M	256	1.0e-3	1.84e+18	0.4880	0.8400	0.1630
93M	256	1.0e-3	3.68e+18	0.5467	0.8607	0.1359
93M	256	1.0e-3	7.37e+18	0.5640	0.8787	0.1176
93M	256	1.0e-3	1.47e+19	0.5967	0.8830	0.1047

C Training Details

Table 3: Architectural details of our five different model sizes.

Model	6.5M	13.5M	24M	45.5M	93M
Model dimension	256	320	384	448	512
Number of encoder layers	3	4	5	7	11
Number of decoder layers	3	4	5	7	11
Number of attention heads	4	5	6	7	8
Head dimension	64	64	64	64	64
MLP dimension	1,024	1,280	1,536	1,792	2,048
Parameters	6.48M	13.40M	24.01M	45.53M	93.08M

Table 4: List of training hyperparameters.

Parameter	Value
Clip value	1.0
Weight decay	0.1
Optimizer	AdamCPR
Beta1	0.9
Beta2	0.98
Eps	1.0e-09
Kappa init param	1,000
Kappa init method	inflection_point
Reg function	l2
Kappa update	1.0
LR warmup steps	5%
LR decay factor	0.01
LR schedule	cosine
Residual dropout	0.1
Attention dropout	0.1
LN eps	1e-5
Param init scale	0.02
Precision	fp32
Max Output Length	256