TOWARDS FOUNDATION MODELS WITH MATHEMATICAL UNDERSTANDING

Peter Belcak
ETH Zürich
8092 Zürich, Switzerland
belcak@ethz.ch

Roger Wattenhofer
ETH Zürich
8092 Zürich, Switzerland
wattenhofer@ethz.ch

ABSTRACT

We investigate the ability of transformer models to build representations of integer sequences that are of utility to tasks where deeper mathematical understanding is needed. To that end, we train BERT-like transformer encoders to assess the impact of individual pre-training tasks on the quality of the resulting model, and evaluate them for sequence classification, continuation, unmasking, complexity prediction, and next sequence-part prediction. We find that the models both outperform benchmark baselines and provide reasonable estimates of the complexity of the mathematical rules behind the sequences.

1 INTRODUCTION

Despite witnessing an abundance of reasoning and computation in their training datasets, foundation models consistently fail at both. This has already begun to be studied, with fine-tuning on structured language introduced for the enhancement of deductive abilities (Tafjord et al., 2020), prompting by algorithm suggested for compensating for algorithmic shortcomings (Zhou et al., 2022), and the setting of foundation models only as translators to formal language proposed for proof discovery (Polu & Sutskever, 2020). Nevertheless, teaching computation to neural networks remains an open challenge, as even architectures tailored to that end fail to consistently yield reliable results Belcak & Wattenhofer (2022a;b),

We investigate what it would take for transformer foundation models to learn an implicit, internal understanding of arithmetics, deductive reasoning, and computation more broadly. A previous inquiry has shown that using number embeddings that have been built in the context of integer sequences while working with natural language can enhance the performance of transformer models across a range of language understanding and processing tasks (Ryskina & Knight, 2021). We stride ahead and consider a setting in which transformer models are asked to operate entirely on integer sequences, hoping to either identify the limits of the architecture or discover new ways of encouraging it to internalise abstract computational concepts.

Recent work in this direction has employed transformer models for example-driven deep symbolic regression (Lample & Charton, 2019; Petersen et al., 2019; d'Ascoli et al., 2022; Kamienny et al., 2022). Branching towards work that is neural end-to-end, Belcak et al. (2022) proposed to investigate the building of neural representations of sequences that mirror their respective governing abstractions well. The paper introduced the Finitary Abstraction Comprehension Toolkit (FACT), a large annotated dataset of integer sequences of both organic and synthetic nature.

In this light, we treat the problem of building neural representations of integer sequences as the limit case of the complementing natural language tasks in which all of the tokens are elements of an integer sequence. We investigate the following questions

- Q1. What is the impact of individual pre-training tasks on the performance of the resulting model across all evaluation tasks?
- Q2. What is the utility of multi-task pre-training when compared to task-specific transformer model training as seen in the FACT benchmark?

Q3. Are pre-trained transformer models capable of effectively understanding deeper concepts behind the integer sequence such as the complexity of the generating rule?

We find that sequence unmasking and classification tasks contribute the most to the building of representations, that multi-task pre-training has a decisive advantage over task-specific training of transformers, and that the pre-trained transformer models are indeed capable of reasonable sequence complexity prediction.

With full reproducibility in mind, we make all our code, training configurations, hyperparameters, and seeds available at *anonymised*. We also make the final pre-trained model readily available for direct use by publishing its weights and training states. We hope that our work will aid future research on learning of algorithmic abstractions by foundation models and help to advance our understanding of the nature and limits of transformer representations.

2 Experimental Setup

For each model instance considered, we first pre-train a network using the tasks of the given experimental configuration, and the fine-tune it for every evaluation tasks of the experiment, adding layers and changing losses where necessary. In case the evaluation task was also used as a pre-training task, we still perform fine-tuning for consistency across experiments.

2.1 Architecture

We use the transformer encoder (Vaswani et al., 2017) with the original sinusoidal positional embedding scheme. In Appendix D, we experiment with different number of transformer layers, attention heads, and neurons per feed-forward layer. We find that the optimal performance was achieved when using 4 transformer layers, 4 attention heads per layer, and 2000 neurons per feed-forward layer.

Our transformer is equipped with regressive unmasking, classification, complexity, and next sequence-part prediction heads simultaneously. When running the ablative experiments of Section 2.5, the head corresponding to the task that is being left our is detached from the transformer. We elaborate on this in Appendix C.

2.2 Data

We use the FACT dataset (Belcak et al., 2022), split into three parts: the training set containing $\frac{9}{10}$ths of all entries, and the validation and test sets containing $\frac{1}{20}$th each. The training set consists only of FACT sequences, whereas the validation and test sets are joined with the $\frac{1}{2}$ of the OEIS dataset pre-selected in line with the criteria of previous work. The dataset is illustrated in Appendix A.

2.3 Input features

For each sequence of the dataset, we consider the first 50 numbers. We design 6 input features per element of the sequence such that they uniformly represent the shapes, magnitudes, and signs of the sequences, resulting in 20 dimensions per number. For the `norm` feature, we normalise each number by the maximum absolute value in the sequence. The resulting sequence of values in the interval $[-1, +1]$ then gives the "shape" of the sequence. For magnitude (feature `log`), we take the base-10 logarithms of 1 plus the absolute value of every number. We write any number in its base-10 positional notation and extract all (at most 15) `digits` into separate input feature dimensions. We also record the `position` of each number within the sequence. The `sign` and `mask` indicators are recorded in separate feature dimensions as well. A detailed overview and an example of this encoding is given in Appendix B.

2.4 Tasks

We considered the sequence unmasking, classification, complexity prediction, and next-sequence part prediction tasks for pre-training. The same tasks were also used for model evaluation, together with sequence similarity, contrastive sequence unmasking, and regressive and contrastive sequence continuation.

Leave-one-out	Task									
	Unmasking								Classification	
	[*sign-acc*]	[*log-MAE*]	[*norm-MAE*]	[*log-naive-RE*]	[*norm-naive-RE*]	[*top-1-RMSE*]	[*top-3-RMSE*]	[*top-5-RMSE*]	[*macro-F1*]	[*micro-F1*]
Unmasking	0.9968	0.0586	0.0480	0.2330	0.2070	1.355	1.178	1.111	0.5969	0.5621
Classification	0.9974	0.0601	0.0401	0.2337	0.1904	**1.289**	**1.112**	**1.051**	0.5972	0.5618
Complexity	**0.9978**	**0.0450**	**0.0383**	**0.2280**	**0.1849**	1.359	1.169	1.095	0.6024	0.5613
NSPP	0.9976	0.0554	0.0410	0.2311	0.1945	1.313	1.134	1.071	0.6016	**0.5646**
with all	0.9954	0.0857	0.0560	0.2470	0.2083	1.316	1.138	1.068	**0.6025**	0.5592
	Complexity		NSPP	Continuation				Similarity		
	[*term-MSE*]	[*depth-MSE*]	[*acc*]	[*log-MSE*]	[*top-1-RMSE*]	[*top-3-RMSE*]	[*top-5-RMSE*]	[*top-1-acc*]	[*top-3-acc*]	[*top-5-acc*]
Unmasking	2.198	4.583	0.9885	0.0162	**1.066**	**0.9165**	**0.8670**	0.0640	0.1387	0.1907
Classification	2.225	**4.548**	0.9882	0.0179	1.099	0.9432	0.8923	0.0657	0.1473	0.1997
Complexity	2.228	4.565	**0.9885**	**0.0125**	1.093	0.9376	0.8839	0.0673	0.1477	0.1993
NSPP	2.208	4.611	0.9863	0.0166	1.149	0.9873	0.9313	0.0640	0.1367	0.1903
with all	**2.177**	4.775	0.9833	0.0162	1.224	1.0680	1.0080	**0.0673**	**0.1507**	**0.2067**

Table 1: The analysis of the impact of individual pre-training tasks on the quality of the resulting model. *acc* denotes accuracy, *MAE* mean absolute error, *naive-RE* the naitve relative error computed as the MAE divided by the ground truth (ignoring NaNs), and *RMSE* root mean squared error. **Emphasis** marks the best performing combination of pre-training tasks in that metric.

We give a detailed overview of these tasks in Appendix E. Let us just note that unmasking, continuation, classification, and next part prediction are typical to language model training, and we employ them in the appropriate adaptation of their original form as seen in natural language processing. Performance on other tasks, especially the contrastive ones, is measured to give an assessment of the quality of the models' embedding spaces. The sequence complexity prediction task is entirely new.

2.4.1 Sequence complexity prediction

The aim of the sequence complexity prediction as a pre-training task is to teach the model to differentiate between sequences representable by rules of differing complexity. We extended the entries of the FACT dataset to provide three values per entry, describing the literal length of the formula used, the depth of its abstract syntax tree (AST), and the number of terminals (leaves) appearing in its AST. We performed regression on every value with mean squared error loss and summed the values to give the total prediction loss. Since the OEIS dataset contains no information about the complexity of a sequence, we ran our evaluation only on FACT sequences. The ASTs are constructed by recursive-descent parsers written for the grammars given by FACT.

2.5 The Impact of Pre-Training Tasks

Table 1 gives the results of the task ablations in our experimental setup. The ablations are performed in leave-one-out fashion: for each ablation, one of the four pre-training tasks is not applied to train the model. The number of samples per batch is adjusted so that the total number of times each sample in the dataset is seen by the model is the same as if all four tasks were used. We make a number of observations:

1. The complexity prediction task causes the highest deterioration in performance on all of regressive sequence unmasking, next sequence-part prediction, and regressive sequence continuation.
2. Somewhat counter-intuitively, the removal of the classification task leads to the highest model performances for the contrastive unmasking. This is unexpected, as the classification task brings in additional information about sequences' adherence to respective categories, which should boost contrastive lookup.
3. The removal of the regressive unmasking task boosts the performance on contrastive continuation, whereas using all tasks together leads to the worst contrastive continuation performance of all ablation runs.
4. Using all pre-training tasks yields the best results for sequence similarity in all metrics.

Observations 2 and 3 seem to suggest that the representations of utility to contrastive lookup tasks are of nature differing fundamentally from those well-suited for regressive and categorisation tasks. Inspecting the drops and increases of the task ablations in Table 1 relative to the results of models trained with all tasks, we find the unmasking pre-training task to be of the highest utility to pre-training for mathematical understanding, followed by classification, next sequence-part prediction, and complexity prediction, in this order. This gives an answer to Q1 of Section 1.

3 The Utility of Pre-Training to Model Understanding

Figure 1 gives the results for our final transformer model, trained on all pre-training tasks considered in our inquiry. We see that it outperforms the previous work in contrastive sequence unmasking, sequence classification, and regressive and contrastive sequence continuation. Further, it achieves solid results in regressive unmasking: it predicts the sign incorrectly 0.08% of the time, its logarithm (in $[1, 16)$) is off by 0.03 on average, and its predictions of the normalised element value (in $[0, 1]$) are off by 0.02 on average.

The one task in which we fail to finish ahead of the benchmarking runs presented in the literature is sequence similarity prediction. As pointed out in Section 2.5, we believe that this is caused by the heavy focus of our pre-training on regressive tasks, and the apparently different nature of neural representations of utility to regressive and contrastive tasks in contrast.

Thus, in answer to Q2, we find that pre-training is of great utility to model performance the tasks considered, even outperforming the best models trained specifically for the given tasks.

Task	Metric	Ours	FACT
Unmasking reg.	sign-acc	**0.9992**	-
	log-MAE	**0.0294**	-
	norm-MAE	**0.0197**	-
Unmasking con.	top-1-RMSE	**0.7470**	3.355–
	top-3-RMSE	**0.3276**	2.69—
	top-5-RMSE	**0.1748**	2.44—
Classification	macro-F1	**0.6205**	0.5665
	micro-F1	**0.5624**	-
Complexity	term-MSE	**2.1820**	-
	depth-MSE	**4.6990**	-
NSPP	accuracy	**0.9922**	0.984–
Similarity	top-1-acc	0.0875	**0.15—**
	top-3-acc	0.1917	**0.39—**
	top-5-acc	0.2333	**0.60—**
Continuation reg.	log-MSE	**0.0075**	0.395–
Continuation con.	top-1-RMSE	**0.4960**	0.847–
	top-3-RMSE	**0.1495**	0.383–
	top-5-RMSE	**0.0678**	0.267–

Figure 1: The results of our final model. "FACT" refers to the performance of the best (usually transformer) model of the FACT benchmark, "con." marks a contrastive task variant, and "reg." marks a regressive variant.

4 Complexity Prediction

The performance of models on the complexity prediction tasks is listed in Table 1 and Figure 1. In complexity prediction, the MSE of 1.48^2 is seen in the prediction of rule terminal length (in $[1, 15]$, and we find that most of the predictions are off by only one or two terminals. We observe that the predictions of the rule depth (in $[1, 14]$) are less accurate (MSE of 2.17^2), but note that the training distribution is logarithmically skewed towards the shallower sequences. Such good performance on the complexity prediction task demonstrates more broadly that transformer networks can indeed learn and understand involved concepts behind the input data, answering Q3.

REFERENCES

Peter Belcak and Roger Wattenhofer. Neural combinatorial logic circuit synthesis from input-output examples. In *2nd Workshop on Math-AI (MATH-AI @ NeurIPS)*, 2022a.

Peter Belcak and Roger Wattenhofer. Periodic extrapolative generalisation in neural networks. In *IEEE Symposium on Deep Learning*, 2022b.

Peter Belcak, Ard Kastrati, Flavio Schenker, and Roger Wattenhofer. Fact: Learning governing abstractions behind integer sequences. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17968–17980. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/72372ec86dd49238900fc0b68bad63f8-Paper-Datasets_and_Benchmarks.pdf.

Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. Deep symbolic regression for recurrent sequences. *arXiv preprint arXiv:2201.04600*, 2022.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532*, 2022.

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

Brenden K Petersen, Mikel Landajuela Larma, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Maria Ryskina and Kevin Knight. Learning mathematical properties of integers. *arXiv preprint arXiv:2109.07230*, 2021.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. Proofwriter: Generating implications, proofs, and abductive statements over natural language. *arXiv preprint arXiv:2012.13048*, 2020.

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.

Hattie Zhou, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. Teaching algorithmic reasoning via in-context learning. *arXiv preprint arXiv:2211.09066*, 2022.

ACKNOWLEDGEMENTS

We thank Flavio Schenker for his work and support with the experimental setup.

A THE JOINT STRUCTURE OF OEIS AND FACT DATASETS

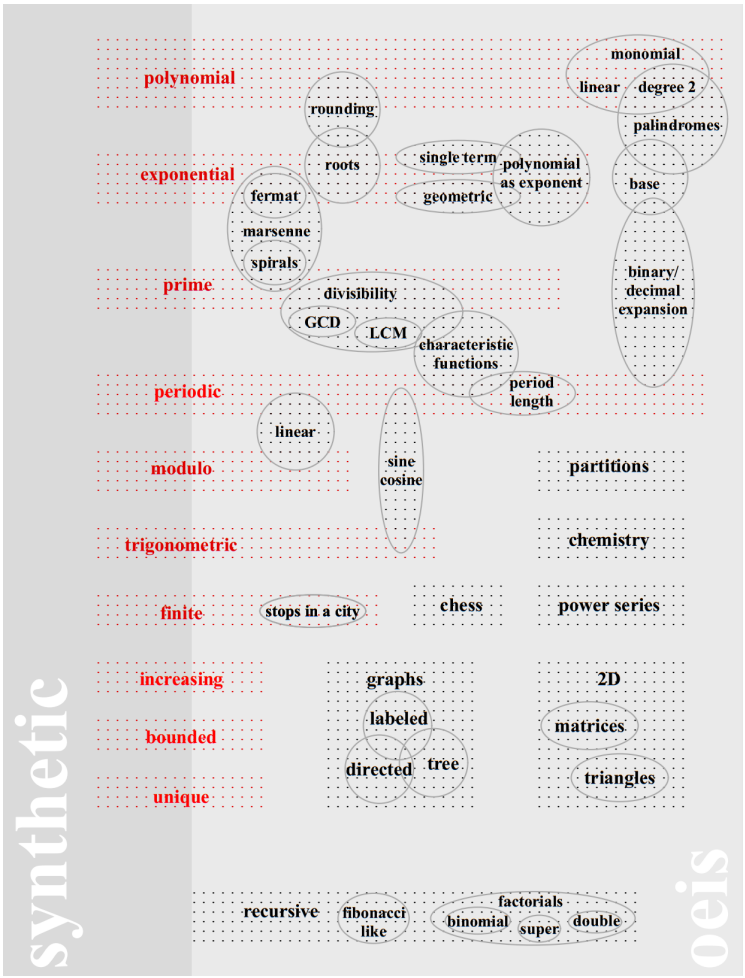

Figure 2: The categories in the OEIS and FACT datasets, joined. Dotted regions represent the main categories identified in the datasets. Ellipses define the sub-categories that emerged from the OEIS processing and annotation effort of Belcak et al. (2022). Red dots mark groups that are augmented with synthetic data.

B INPUT FEATURES

Table 2 describes how the values are computed and gives a short description for every feature.

Feature	Values	Description
Position	$\{0, 1, ..., 49\}$	Index of the number in the sequence.
Sign	$\{0, 1\}$	1 if the number is negative, else 0.
normalised	$\lvert n\rvert \div \max \lvert \underline{s}\rvert$	normalised number.
Log	$\log_{10} 1 + \lvert n\rvert$	The logarithm of each number.
Digits	$\{1, 2, .., 9\}$	Base 10 digits in positional notation.
Mask	$\{0, 1\}$	1 if the number is masked, else 0.

Table 2: Overview of the input features used in our models.

In equation B an example of a possible feature extraction is given. The example sequence is of length 4 instead of 50, and 9 features (4 digits positions instead of 15) were used.

$$[302 \quad -604 \quad 1208 \quad masked]^\top \Rightarrow \begin{bmatrix} 0 & 0 & 0.25 & 2.48 & 2 & 0 & 3 & 0 & 0 \\ 1 & 1 & 0.50 & 2.78 & 4 & 0 & 6 & 0 & 0 \\ 2 & 0 & 1.00 & 3.08 & 8 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0.00 & 0.00 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The tensor of our final dataset is in the shape `(3249506, 50, 20)`.

C Model and Training Details

An illustration of our model can be seen in Figure 3. We use a base encoder, which we supply with different output heads for every pre-training task. Each output head is a multi-layer perceptron (MLP) tailored to the task at hand. We describe every output head in detail in Appendix C.1.1. For downstream tasks, we take the base encoder (marked in white) and transfer it to our final model for fine-tuning, where we append different MLPs for each task again.

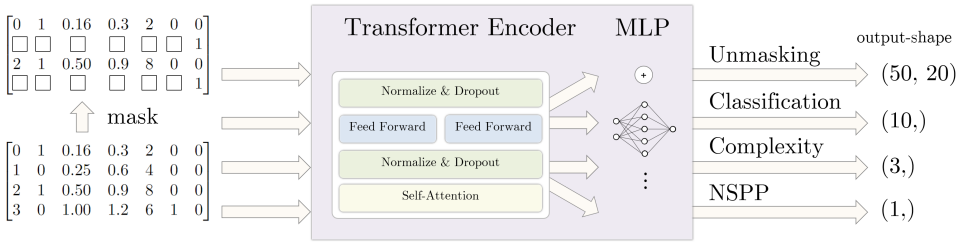

Figure 3: Our model architecture. The basis transformer encoder with multi-layer perceptrons attached for each pre-training task.

C.1 Neural network

As our base encoder we use a transformer encoder, that follows the architecture of Vaswani et al. (2017). Each multi-head attention unit consists of four attention heads and the output dimension of the feed-forward layers is 2000. The positional encoding is done with the original sinusoidal approach. Further parameters used, are listed in Table 4.

C.1.1 Model heads

In Table 3 we describe every MLP which was attached to our base encoder, first during the regime and model inference, and secondly during fine-tuning. We provide the dimensions in bracket notation with each entry representing the dimension of one densely connected hidden layer. The last entry is the output dimension.

Task version	inference	fine-tuning
Unmasking reg.	[2000, 1000]	[2000, 4000, 2000]
Unmasking con.	[2000, 4000, 8000]	[2000, 4000, 8000]
Classification	[2000, 10]	[200, 100, 10]
Complexity	[2000, 3]	[200, 100, 3]
NSPP	[2000, 1]	[200, 100, 20, 10, 2, 1]
Continuation reg.	[2000, 4000, 1]	[2000, 4000, 1]
Continuation con.	[2000, 4000, 8000]	[2000, 4000, 8000]
Similarity	[2000, 50]	[100, 20]

Table 3: All MLP model heads in bracket notation.

C.2 General parameters

Table 4 lists all general and additional parameters we finally choose, for the optimization and fine-tuning described in the previous chapters.

C.3 Flexible contrastive loss

In some tasks we used contrastive learning to achieve a desired embedding space $\mathcal{Z}$, namely sequence similarity, sequence continuation multi-shot and contrastive sequence unmasking. There, we went for

Batch size	120
Base random seed	22082015
Optimizer	Adam
Scheduler	exponential LR-decay
Learning rate	1e-4
Learning rate decay	0.99
Transformer dropout probability	0.10
L2-weight-decay regularization	1e-4
Classification confidence threshold	0.55
Unmasking mask probability	0.25
Continuation mask length	5
Unmasking margin	10
Continuation margin	1
Similarity margin	1

Table 4: An overview of the general training parameters.

a contrastive loss described with the following formula:

$$
\begin{aligned}
d_a &= max\big(\alpha - d, 0\big)^2 \\
d_n &= d^2 \\
\mathcal{L} &= (1-\lambda)d_a + \lambda d_n
\end{aligned}
$$

where d is the euclidean distance between two sequences in the embedding space and α is the margin-distance which penalises dissimilar pairs only if their distance d is inside its radius. The goal of this loss is to embed similar sequences near each other in terms of the euclidean distance and different ones further away. The parameter λ functions as a measurement of similarity. In each contrastive task we define this measurement with a slight variation. In sequence similarity, λ is the indicator function between two classes. In sequence continuation, λ is the fraction between the first n similar numbers of two sequences and its total length, whereas in sequence unmasking λ is the fraction of masked entries in a sequence paired with its unmasked counterpart. With this approach we seek to build an embedding space that learns to differentiate different sequences, according to our predefined tasks.

C.4 Computational resources and code

We trained each of our models a single Nvidia Titan Xp. Experiments (pre-training instances or evaluations on a specific task) lasted on average between 2.5 and 5 hours each. No experiment lasted more than 37 hours.

D ARCHITECTURAL ABLATIONS

Dimensions			Task										
			Unmasking					Classification		Complexity		NSPP	
attention-heads	transformer-blocks	feed-forward	[*sign-Acc.*]	[*log-MAE*]	[*norm-MAE*]	[*log-RE*]	[*norm-RE*]	[*macro-F1*]	[*micro-F1*]	[*terminal-MSE*]	[*depth-MSE*]	[*Acc.*]	
2	2	1000	**0.9991**	0.06587	0.03762	0.2337	0.1915	0.6065	0.5606	1.947	4.541	0.9873	
2	2	2000	0.9980	0.06288	0.04131	0.2196	0.1921	0.6033	0.5600	2.120	4.221	**0.9933**	
2	2	4000	0.9983	0.04947	0.03828	0.2285	0.1994	0.6001	0.5585	2.361	4.166	0.9847	
2	4	1000	0.9987	0.04899	0.04450	0.2405	0.1865	0.6041	0.5700	**1.822**	4.649	0.9813	
2	4	2000	0.9976	0.05403	0.04100	0.2210	0.1825	0.6067	0.5655	2.178	4.600	0.9900	
2	4	4000	0.9978	0.07025	0.05045	0.2308	0.2036	**0.6206**	**0.5760**	2.120	4.323	0.9867	
2	6	1000	0.9978	0.06254	0.05327	0.2397	0.2101	0.6025	0.5544	2.527	3.978	0.9873	
2	6	2000	0.9964	0.04697	0.04460	0.2239	0.1834	0.6100	0.5604	2.436	3.728	0.9893	
2	6	4000	0.9971	0.04786	0.04426	0.2270	0.1836	0.5901	0.5420	2.031	4.467	0.9880	
4	2	1000	0.9986	0.06819	0.03802	0.2375	0.1930	0.6111	0.5626	1.918	4.590	0.9867	
4	2	2000	0.9979	0.07201	0.04110	0.2231	0.1896	0.6026	0.5582	2.133	4.300	0.9920	
4	2	4000	0.9983	0.05135	0.04066	0.2286	0.2020	0.6011	0.5590	2.389	4.148	0.9853	
4	4	1000	0.9985	0.05133	0.04054	0.2380	0.1870	0.6092	0.5743	1.831	4.694	0.9827	
4	4	2000	0.9974	0.04672	0.03897	0.2194	0.1782	0.6073	0.5665	2.191	4.644	0.9913	
4	4	4000	0.9977	0.07548	0.03914	0.2271	0.1931	0.6099	0.5703	2.127	4.269	0.9873	
4	6	1000	0.9966	0.05277	0.04056	0.2345	0.1983	0.6059	0.5555	2.519	3.945	0.9873	
4	6	2000	0.9967	**0.04372**	0.04469	0.2220	0.1884	0.6110	0.5628	2.438	**3.707**	0.9893	
4	6	4000	0.9974	0.06535	0.04811	0.2322	0.1917	0.5923	0.5454	2.030	4.403	0.9900	
6	2	1000	0.9968	0.06140	0.03801	0.2364	0.1851	0.6064	0.5606	2.349	4.638	0.9887	
6	2	2000	0.9968	0.05189	0.03916	0.2310	0.1721	0.6113	0.5635	2.312	4.076	0.9880	
6	2	4000	0.9981	0.05053	0.04511	0.2185	0.1935	0.6095	0.5659	2.040	4.017	0.9893	
6	4	1000	0.9986	0.05042	0.04088	0.2262	0.2050	0.6035	0.5629	2.454	5.192	0.9827	
6	4	2000	0.9974	0.05504	**0.03672**	0.2344	0.1723	0.6087	0.5664	2.088	4.304	0.9887	
6	4	4000	0.9978	0.06500	0.03864	0.2290	0.1853	0.5960	0.5503	2.183	4.366	0.9880	
6	6	1000	0.9967	0.04422	0.03815	**0.2000**	**0.1672**	0.5948	0.5500	2.328	4.433	0.9853	
6	6	2000	0.9978	0.07446	0.05288	0.2469	0.2095	0.6039	0.5580	2.067	4.259	0.9880	
6	6	4000	0.9988	0.06930	0.04423	0.2472	0.1908	0.6085	0.5542	2.145	4.164	0.9853	

Table 5: Architectural ablations of the model dimensions. *Acc.* denotes means accuracy, *MAE* mean-absolute-error, *RE* relative error, and *RMSE* root mean squared error. **Emphasis** marks the best performing model in that metric.

E OVERVIEW OF PRE-TRAINING AND EVALUATION TASKS

Our initial experimentation considered all FACT benchmark tasks together with regressive unmasking and complexity prediction as candidate pre-training tasks. Regressive and contrastive continuation, contrastive unmasking, and similarity tasks did not improve the performance of the pre-trained model on the downstream tasks. In fact, the contrastive tasks had tendency to worsen the performance. Hence, only sequence unmasking, classification, complexity prediction, and next sequence-part prediction were considered for pre-training. All tasks were used for model evaluation.

E.1 SEQUENCE UNMASKING

Typical to transformer pre-training in natural language processing, the unmasking task has been proved to excel at encouraging the formation of contextualised embeddings for the individual elements of sequential inputs. The objective of the task is to find the elements that have been masked out in the inputs (and specially marked as such) from the remaining context. In our setup, we mark each token as masked and set its input features to zero, with probability 0.25. In contrast to the classification configuration common in natural language processing, our task is regressive – the model is tasked to

output the individual features of every masked token directly, and not to predict which element from a finite vocabulary was masked. We tailored our loss function to learn the shapes, magnitudes, and signs of the sequences by having each of these aspects contribute to the loss. For evaluation, we also consider unmasking by contrastive lookup as in the FACT benchmark.

E.2 SEQUENCE CLASSIFICATION

The classification task incentivises the transformer to learn traits that are common to classes of sequences. Sequence categories of the FACT and OEIS datasets are non-exclusive and imbalanced. We train with binary cross-entropy loss on each dimension and use use the macro- and micro-averaged F1 scores as evaluation metrics.

E.3 SEQUENCE COMPLEXITY PREDICTION

The aim of the sequence complexity prediction as a pre-training task is to teach the model to differentiate between sequences representable by rules of differing complexity. We extended the entries of the FACT dataset to provide three values per entry, describing the literal length of the formula used, the depth of its abstract syntax tree (AST), and the number of terminals (leaves) appearing in its AST. We performed regression on every value with mean squared error loss and summed the values to give the total prediction loss. Since the OEIS dataset contains no information about the complexity of a sequence, we ran our evaluation only on FACT sequences. The ASTs are constructed by recursive-descent parsers constructed for the grammars given by FACT.

E.4 NEXT SEQUENCE-PART PREDICTION

Given two sub-sequences $\underline{s}_1, \underline{s}_2$, the objective of the next sequence-part prediction task (NSPP) is to determine if the sub-sequence $\underline{s}_2$ is a valid continuation of $\underline{s}_1$ or not. We train with binary cross-entropy loss and evaluate performance by binary accuracy.

E.5 SEQUENCE SIMILARITY

The objective of the sequence similarity task is to group sequences that are similar in their type (e.g. are both polynomial) and properties (e.g. both are periodic and bounded) together. For training, we equip the frozen transformer with an additional linear layer that matches the flattened size of the transformer's output and fine-tune it in a contrastive fashion. For evaluation, we randomly select $k \cdot m$ candidates for a sequence $\underline{s}$ by sampling k sequences from each of the m categories, and then order them by their euclidean distance to $\underline{s}$. As metric we use top-k-accuracy, calculated as the proportion of the top-k candidates that are in the same category as $\underline{s}$.

E.6 SEQUENCE CONTINUATION

Sequence continuation is to suggest the entries that are to follow a given sequence prefix $\underline{s}$. This task is *extrapolative* in its nature, and is meant to challenge model understanding beyond making a binary decision between externally provided suggestions. We distinguish two sub-types of this task: For the *single-shot* continuation, we predict a single candidate for the next entry, while in the *multi-shot* setup, the entire continuation of the sequence is to be predicted.