Deep Symbolic Regression for Recurrent Sequences

Stéphane d'Ascoli [* 1 2] **Pierre-Alexandre Kamienny** [* 2 3] **Guillaume Lample** [2] **François Charton** [2]

Abstract

Symbolic regression, i.e. predicting a function from the observation of its values, is well-known to be a challenging task. In this paper, we train Transformers to infer the function or recurrence relation underlying sequences of integers or floats, a typical task in human IQ tests which has hardly been tackled in the machine learning literature. We evaluate our integer model on a subset of OEIS sequences, and show that it outperforms built-in Mathematica functions for recurrence prediction. We also demonstrate that our float model is able to yield informative approximations of out-of-vocabulary functions and constants, e.g. $\text{bessel0}(x) \approx \frac{\sin(x)+\cos(x)}{\sqrt{\pi x}}$ and $1.644934 \approx \pi^2/6$.

1. Introduction

Given the sequence [1,2,4,7,11,16], what is the next term? Humans usually solve such riddles by noticing patterns in the sequence. In the easiest cases, one can spot a function: [1,4,9,16,25] are the first five squares, so the n-th term in the series is $u_n = n^2$, and the next one is 36. Most often however, we look for relations between successive terms: in the sequence [1,2,4,7,11,16], the differences between successive values are 1, 2, 3, 4, and 5, which makes it likely that the next term will be $16 + 6 = 22$. Mathematically, we are inferring the recurrence relation $u_n = u_{n-1} + n$, with $u_0 = 1$.

In all cases, we handle such problems as symbolic regressions: starting from a sequence of numbers, we try to discover a function or a recurrence relation that they satisfy, and use it to predict the next terms. This can lead to very challenging problems as the complexity of the unknown recurrence relation $u_n = f(n, \{u_i\}_{i<n})$ increases,

e.g. $u_n = \tan^{-1}(u_{n-3}) \exp(\cos(n^2))$.

In this paper, we train neural networks to infer the recurrence relation f from the observation of the first terms of the sequence. The majority of studies in machine learning for symbolic regression have focused on non-recurrent functions, i.e. expressions of the form $y = f(x)$. Recurrence relations provide a more general setting, which gives a deeper account of the underlying process: if the sequence corresponds to successive time steps, the recurrence relation is a discrete time differential equation for the system considered.

1.1. Contributions

We show that transformers can learn to infer a recurrence relation from the observation of the first terms of a sequence. We consider both sequences of integers and floats, and train our models on a large set of synthetic examples.

We first demonstrate that our symbolic regression model can predict complex recurrence relations that were not seen during training. We also show that those recurrence relations can be used to extrapolate the next terms of the sequence with better precision than a "numeric" model of similar architecture, trained specifically for extrapolation.

We then test the out-of-domain generalization of our models. On a subset of the Online Encyclopedia of Integer Sequences, our integer models outperform built-in Mathematica functions, both for sequence extrapolation and recurrence prediction, see Table 1 for some examples. We also show that our symbolic float model is capable of predicting approximate expressions for out-of-vocabulary functions and constants (e.g. $\text{bessel0}(x) \approx \frac{\sin(x)+\cos(x)}{\sqrt{\pi x}}$ and $1.644934 \approx \pi^2/6$), see Tables 2, 3 for more examples.

We conclude by discussing potential limitations of our models and future directions.

1.2. Related work

AI for maths The use of neural networks for mathematics has recently gained attention: several works have demonstrated their surprising reasoning abilities (Saxton et al., 2019; Cobbe et al., 2021), and have even sparked some interesting mathematical discoveries (Davies et al., 2021). In

---

*Equal contribution [1]Department of Physics, École Normale Supérieure, Paris [2]Meta AI, Paris [3]Laboratoire d'Informatique de Paris 6, Sorbonne Université, Paris. Correspondence to: Stéphane d'Ascoli <stephane.dascoli@ens.fr>.

OEIS	Description	First terms	Predicted recurrence
A000792	$a(n) = \max\{(n-i)a(i), i < n\}$	1, 1, 2, 3, 4, 6, 9, 12, 18, 27	$u_n = u_{n-1} + u_{n-3} - u_{n-1}\%u_{n-3}$
A000855	Final two digits of 2^n	1, 2, 4, 8, 16, 32, 64, 28, 56, 12	$u_n = (2u_{n-1})\%100$
A006257	Josephus sequence	0, 1, 1, 3, 1, 3, 5, 7, 1, 3	$u_n = (u_{n-1} + n)\%(n-1) - 1$
A008954	Final digit of $n(n+1)/2$	0, 1, 3, 6, 0, 5, 1, 8, 6, 5	$u_n = (u_{n-1} + n)\%10$
A026741	$a(n) = n$ if n odd, $n/2$ if n even	0, 1, 1, 3, 2, 5, 3, 7, 4, 9	$u_n = u_{n-2} + n//(u_{n-1}+1)$
A035327	n binary, switch 0's and 1's, then decimal	1, 0, 1, 0, 3, 2, 1, 0, 7, 6	$u_n = (u_{n-1} - n)\%(n-1)$
A062050	n-th chunk contains numbers $1, ..., 2^n$	1, 1, 2, 1, 2, 3, 4, 1, 2, 3	$u_n = (n\%(n - u_{n-1})) + 1$
A074062	Reflected Pentanacci numbers	5, -1, -1, -1, -1, 9, -7, -1, -1, -1	$u_n = 2u_{n-5} - u_{n-6}$

Table 1. **Our integer model yields exact recurrence relations on a variety of interesting OEIS sequences.** Predictions are based on observing the first 25 terms of each sequence.

Constant	Approximation	Rel. error
0.3333	$(3 + \exp(-6))^{-1}$	10^{-5}
0.33333	$1/3$	10^{-5}
3.1415	$2\arctan(\exp(10))$	10^{-7}
3.14159	π	10^{-7}
1.6449	$1/\arctan(\exp(4))$	10^{-7}
1.64493	$\pi^2/6$	10^{-7}
0.123456789	$10/9^2$	10^{-9}
0.987654321	$1 - (1/9)^2$	10^{-11}

Table 2. **Our float model learns to approximate out-of-vocabulary constants with its own vocabulary.** We obtain the approximation of each constant C by feeding our model the 25 first terms of $u_n = Cn$.

Expression u_n	Approximation $\hat{u}_n$	Comment
$\operatorname{arcsinh}(n)$	$\log(n + \sqrt{n^2 + 1})$	Exact
$\operatorname{arccosh}(n)$	$\log(n + \sqrt{n^2 - 1})$	Exact
$\operatorname{arctanh}(1/n)$	$\frac{1}{2}\log(1 + 2/n)$	Asymptotic
$\operatorname{catalan}(n)$	$u_{n-1}(4 - 6/n)$	Asymptotic
$\operatorname{dawson}(n)$	$\frac{n}{2n^2 - u_{n-1} - 1}$	Asymptotic
$j0(n)$ (Bessel)	$\frac{\sin(n) + \cos(n)}{\sqrt{\pi n}}$	Asymptotic
$i0(n)$ (mod. Bessel)	$\frac{e^n}{\sqrt{2\pi n}}$	Asymptotic

Table 3. **Our float model learns to approximate out-of-vocabulary functions with its own vocabulary.** For simple functions, our model predicts an exact expression in terms of its operators; for complex functions which cannot be expressed, our model manages to predict the first order of the asymptotic expansion.

particular, four types of tasks have recently been tackled in the deep learning literature.

First, converting a symbolic expression to another symbolic expression. Direct examples are integration and differentation (Lample & Charton, 2019), expression simplification (Allamanis et al., 2017) or equation solving (Arabshahi et al., 2018). Second, converting a symbolic expression to numerical data. This includes predicting quantitative properties of a mathematical structure, for example the local stability of a differential system (Charton et al., 2020). Third, converting numerical data to numerical data using mathematical rules. Examples range from learning basic arithmetics (Kaiser & Sutskever, 2015; Trask et al., 2018) to linear algebra (Charton, 2021). Fourth, converting numerical data to a symbolic expression: this is the framework of symbolic regression, which we will be focusing on.

Symbolic regression Two types of approaches exist for symbolic regression, which one could name selection-based and pretraining-based.

In selection-based approaches, we only have access to one set of observations: the values of the function we are trying

to infer. Typical examples of this approach are evolutionnary programs, which have long been the *de facto* standard for symbolic regression (Augusto & Barbosa, 2000; Schmidt & Lipson, 2009; Murari et al., 2014; McKay et al., 1995), despite their computational cost and limited performance. More recently, neural networks were used following this approach (Sahoo et al., 2018; Kim et al., 2020; Petersen et al., 2019).

In pretraining-based approaches, we train neural networks on a large dataset containing observations from many different function (Biggio et al., 2021; Valipour et al., 2021), hoping that the model can generalize to new expressions. Although the pretraining is computationally expensive, inference is much quicker as one simply needs to perform a forward pass, rather than search through a set of functions. In this work, we choose this approach.

Recurrent sequences All studies on symbolic regression cited above consider the task of learning non-recurrent functions from their values at a set of points. Our contribution is, to the best of our knowledge, the first to tackle the setup of recurrent expressions. Although this includes non-recurrent

expressions as a special case, one slight restriction is that the inputs need to be sampled uniformly. Hence, instead of feeding the model with input-output pairs $\{(x_i, y_i)\}$, we only need to feed it the terms of the sequence $\{u_i\}$. Another important difference is that order of these terms matters, hence permutation invariant representations (Valipour et al., 2021) cannot be used.

Integer sequences, in particular those from the Online Encyclopedia of Integer Sequences (OEIS) (Sloane, 2007), have been studied using machine learning methods in a few recent papers. Wu (2018) trains various classifiers to predict the label of OEIS sequences, such as "interesting", "easy", or "multiplicative". Ryskina & Knight (2021) use embeddings trained on OEIS to investigate the mathematical properties of integers. Ragni & Klein (2011) use fully connected networks to predict the next term of OEIS sequences (a numeric rather than symbolic regression task). Nam et al. (2018) investigate different architectures (most of them recurrent networks) for digit-level numeric regression on sequences such as Fibonacci, demonstrating the difficulty of the task.

2. Methods

Broadly speaking, we want to solve the following problem: given a sequence of n points $\{u_0, \ldots, u_{n-1}\}$, find a function f such that for any $i \in \mathbb{N}$, $u_i = f(i, u_{i-1}, \ldots, u_{i-d})$, where d is the recursion degree. Since we cannot evaluate at an infinite number of points, we declare that a function f is a solution of the problem if, given the first n_{input} terms in the sequence, it can predict the next n_{pred} following ones.

Under this form, the problem is underdetermined: given a finite number of input terms, an infinite number of recurrence relations exist. However in practice, one would like the model to give us the simplest solution possible, following Occam's principle. This is generally ensured by the fact that simple expressions are more likely to be generated during training.

Integer vs. float We consider two settings for the numbers in the sequences: integers and floats. For integer sequences, the recurrence formula only uses operators which are closed in $\mathbb{Z}$ (e.g. $+$, $\times$, abs, modulo and integer division). For float sequences, additional operators and functions are allowed, such as real division, $\exp$ and $\cos$ (see Table 4 for the list of all used operators). These two setups are interesting for different reasons. Integer sequences are an area of strong interest in mathematics, particular for their relation with arithmetics. The float setup is interesting to see how our model can generalize to a larger set of operators, which provides more challenging problems.

Symbolic vs. numeric We consider two tasks: symbolic regression and numeric regression. In symbolic regression, the model is tasked to predict the recurrence relation the sequence was generated from. At test time, this recurrence relation is evaluated by how well it approximates the n_{pred} following terms in the sequence.

In numeric regression, is tasked to directly predict the values of the n_{pred} following terms, rather than the underlying recurrence relation. At test time, the model predictions are compared with the true values of the sequence.

	Integer	Float
Unary	abs, sqr, sign, relu	abs, sqr, sqrt, inv, log, exp, sin, cos, tan, atan
Binary	add, sub, mul, intdiv, mod	add, sub, mul, div

Table 4. **Operators used in our generators.**

2.1. Data generation

All training examples are created by randomly generating a recurrence relation, randomly sampling its initial terms, then computing the next terms using the recurrence relation. More specifically, we use the steps below:

1. Sample the **number of operators** o between 1 and o_{max}, and build a unary-binary tree with o nodes, as described in (Lample & Charton, 2019). The number of operators determines the difficulty of the expression.

2. Sample the **nodes** of the tree from the list of operators in Table 4. Note that the float case uses more operators than the integer case, which makes the task more challenging by expanding the problem space.

3. Sample the **recurrence degree** d between 1 and d_{max}, which defines the recurrence depth: for example, a degree of 2 means that u_{n+1} depends on u_n and u_{n-1}.

4. Sample the **leaves** of the tree: either a constant, with probability p_{const}, or the current index n, with probability p_n, or one of the previous terms of the sequence u_{n-i}, with $i \in [1, d]$, with probability p_{var}.

5. Recalculate the true recurrence degree d considering the deepest leaf u_{n-i} sampled during the previous step, then sample d **initial terms** from a random distribution $\mathcal{P}$.

6. Sample l between l_{min} and l_{max} and compute the next l terms of the sequence using the initial conditions. The total **sequence length** is hence $n_{input} = d_{\text{eff}} + l$.

We provide the values of the parameters of the generator in Table 5. Note that in the last step, we interrupt the computation if we encounter a term larger than 10^{100}, or outside the range of one of the operators: for example, a division by zero, or a negative square root.

Parameter	Description	Value
d_{max}	Max degree	6
o_{max}	Max number of operators	10
l_{min}	Min length	5
l_{max}	Max length	30
p_{const}	Prob of constant leaf	1/3
p_{index}	Prob of index leaf	1/3
p_{var}	Prob of variable leaf	1/3
$\mathcal{P}$	Distrib of first terms	$\mathcal{U}(-10, 10)$

Table 5. **Hyperparameters of our generator.**

Limitations   One drawback of our approach is poor out-of-distribution generalization of deep neural networks, demonstrated in a similar context to ours by (Charton et al., 2020): when shown an example which is impossible to be generated by the procedure described above, the model will inevitably fail. For example, as shown in App. B, our model performs poorly when the initial terms of the sequence are sampled outside their usual range, which is chosen to be $[-10, 10]$ in these experiments. One easy fix would be to increase this range, but this would only shift the problem.

2.2. Encodings

Model inputs are sequences of integers or floats. The outputs are recurrence relations for the symbolic task, and sequences of numbers for the numeric task. To be processed by transformers, inputs and outputs must be represented as sequences of tokens from a fixed vocabulary. To this effect, we use the encodings presented below.

Integers   We encode integers using their base b representation, as a sequence of $1 + \lceil \log_b |x| \rceil$ tokens: a sign (which also serves as a sequence delimiter) followed by $\lceil \log_b |x| \rceil$ digits from 0 to $b-1$, for a total vocabulary of $b+2$ tokens. For instance, $x = -325$ is represented in base $b = 10$ as the sequence `[-, 3, 2, 5]`, and in base $b = 30$ as `[-, 10, 25]`. Choosing b involves a tradeoff between the length of the sequences fed to the Transformer and the vocabulary size of the embedding layer. We choose $b = 10,000$ and limit integers in our sequences to absolute values below 10^{100}, for a maximum of 26 tokens per integer, and a vocabulary of order 10^4 words.

Floats   Following Charton (2021), we represent float numbers in base 10 floating-point notation, round them to four

significant digits, and encode them as sequences of 3 tokens: their sign, mantissa (between `0` and `9999`), and exponent (from `E-100` to `E100`)[1]. For instance, $1/3$ is encoded as `[+, 3333, E-4]`. Again, the vocabulary is of order 10^4 words.

For all operations in floating-point representation, precision is limited to the length of the mantissa. In particular, when summing elements with different magnitudes, subdominant terms may be rounded away. Partly due to this effect, when approximating complex functions, our symbolic model typically only predicts the largest terms in its asymptotic expansion, as shown in Table 3. We discuss two methods for increasing precision when needed in Section E of the Appendix.

Recurrence relations   To represent mathematical trees as sequences, we enumerate the trees in prefix order, i.e. direct Polish notation, and encode each node as a single autonomous token. For instance, the expression $\cos(3x)$ is encoded as `[cos, mul, 3, x]`.

Note that the generation procedure implies that the recurrence relation is not simplified (i.e. expressions like $1 + u_{n-1} - 1$ can be generated). We tried simplifying them using Sympy before the encoding step (see Appendix C), but this slows down generation without any benefit on the performance of our models, which turn out to be surprisingly insensitive to the syntax of the formulas.

2.3. Experimental details

Similarly to Lample & Charton (2019), we use a simple Transformer architecture (Vaswani et al., 2017) with 8 hidden layers, 8 attention heads and an embedding dimension of 512 both for the encoder and decoder.

Training and evaluation   The tokens generated by the model are supervised via a cross-entropy loss. We use the Adam optimizer, warming up the learning rate from 10^{-7} to 2.10^{-4} over the first 10,000 steps, then decaying it as the inverse square root of the number of steps, following (Vaswani et al., 2017). We train each model for a minimum of 250 epochs, each epoch containing 5M equations in batches of 512. On 16 GPU with Volta architecture and 32GB memory, one epoch is processed in about an hour.

After each epoch, we evaluate the in-distribution performance of our models on a held-out dataset of 10,000 equations. Unless specified otherwise, we generate expressions using greedy decoding. Note that nothing prevents the symbolic model from generating an invalid expression such as `[add,1,mul,2]`. These mistakes, counted as invalid predictions, tend to be very rare: in models trained for more

---

[1]By convention, we represent zero as `[+, 0, E0]`.

than a few epochs, they occur in less than 0.1% of the test cases.

Hypothesis ranking   To predict recurrence relations for OEIS sequences (Section 4.1), or the examples displayed in Tables 1,2,3, we used a beam size of 10. Usually, hypotheses in the beam are ranked according to their log-likelihood, normalized by the sequence length. For our symbolic model, we can do much better, by ranking beam hypotheses according to how well they approximate the initial terms of the original sequence. Specifically, given a recurrence relation of degree d, we recompute for each hypothesis the n_{input} first terms in the sequence (using the first d terms of the original sequence as initial conditions), and rank the candidates according to how well they approximate these terms. This ranking procedure is impossible for the numerical model, which can only perform extrapolation.

In some sense, this relates our method to evolutionary algorithms, which use the input terms for candidate selection. Yet, as shown by the failure modes presented in Section G of the Appendix, our model is less prone to overfitting since the candidates are not directly chosen by minimizing a loss on the input terms.

3. In-domain generalization

We begin by probing the in-domain accuracy of our model, i.e. its ability to generalize to unseen sequences generated with the same procedure as the training data. As discussed in Section D of the Appendix, the diversity of mathematical expressions and the random sampling of the initial terms ensures that almost all the examples presented at test time have not been seen during training: one cannot attribute the generalization to mere memorization.

Prediction accuracy   Due to the large number of equivalent ways one can represent a mathematical expression, one cannot directly evaluate the accuracy by comparing (token by token) the predicted recurrence relation to the ground truth. Instead, we use the predicted expression to compute the next n_{pred} terms of the sequence $\{\hat{u}_i\}$, and compare them with those computed from the ground truth, $\{u_i\}$. The prediction accuracy is then defined as:

$$acc(n_{pred}, \tau) = \mathbb{P}\left(\max_{1 \leq i \leq n_{pred}} \left|\frac{\hat{u}_i - u_i}{u_i}\right| \leq \tau\right) \quad (1)$$

By choosing a small enough $\tau \geq 0$ and a large enough n_{pred}, one can ensure that the predicted formula matches the true formula. In the float setup, $\tau = 0$ must be avoided for two reasons: (i) equivalent solutions represented by different expressions may evaluate differently because due to finite machine precision, (ii) setting $\tau = 0$ would penalize

Model	Integer		Float	
	$n_{op} \leq 5$	$n_{op} \leq 10$	$n_{op} \leq 5$	$n_{op} \leq 10$
Symbolic	**92.7**	**78.4**	**74.2**	**43.3**
Numeric	83.6	70.3	45.6	29.0

Table 6. **Average in-distribution accuracies of our models.** We set $\tau = 10^{-10}$ and $n_{pred} = 10$.

the model for ignoring sub-dominant terms which are indetectable due to the finite precision encodings. Hence, we select $\tau = 10^{-10}$ and $n_{pred} = 10$ unless specified otherwise[2]. Occasionally, we will consider larger values of τ, to assess the ability of our model to provide approximate expressions.

Results   The average in-distribution accuracies of our models are reported in Table 6 with $\tau = 10^{-10}$ and $n_{pred} = 10$. Although the float setup is significantly harder that the integer setup, our symbolic model reaches good accuracies in both cases. In comparison, the numeric model obtains worse results, particularly in the float setup. The interested reader may find additional training curves, attention maps and visualizations of the model predictions in Appendix G.

Ablation over the evaluation metric   The two first panels of Figure 1 illustrate how the accuracy changes as we vary the tolerance level τ and the number of predictions n_{pred}. The first important observation is that the symbolic model performs much better than the numeric model at low tolerance. At high tolerance, it still keeps an advantage in the integer setup, and performs similarly in the float setup. This demonstrates the advantage of a symbolic approach for high-precision predictions.

Second, we see that the accuracy of both models degrades as we increase the number of predictions n_{pred}, as one could expect. However, the decrease is less important for the symbolic model, especially in the float setup where the curve is essentially flat. This demonstrates another strong advantage of the symbolic approach: once it has found the correct formula, it can predict the whole sequence, whereas the precision of the numeric model deteriorates as it extrapolates further.

Ablation over the example difficulty   The three last panels of Figure 1 decompose the accuracy of our two models along three factors of difficulty: the number of operators o[3], the recurrence degree d and the sequence length n_{input} (see

---

[2]For the float numeric model, which can only predict values up to finite precision ϵ, we round the values of target function to the same precision. This explains the plateau of the accuracy at $\tau < \epsilon$ in Figure 1.

[3]Since expressions are not simplified, o may be overestimated.
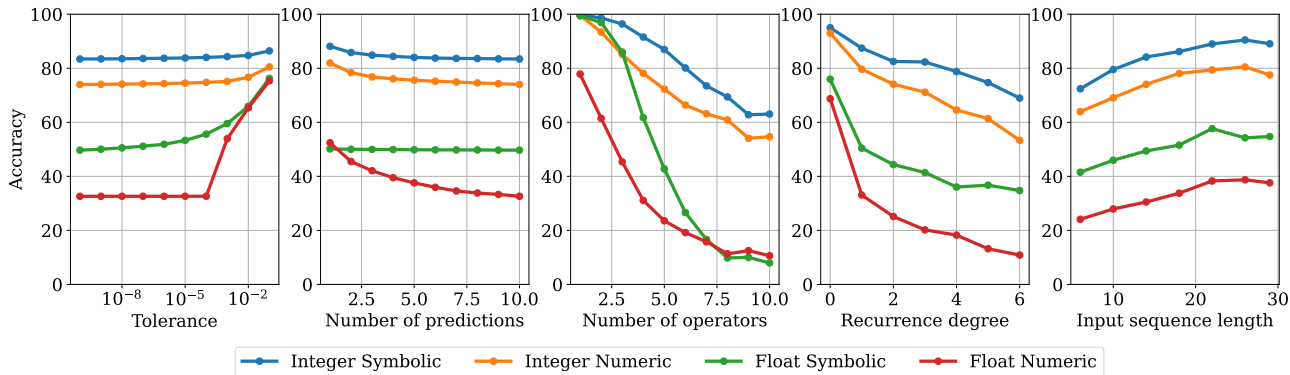
Figure 1. **The symbolic model extrapolates further and with higher precision than the numeric model.** From left to right, we vary the tolerance τ, the number of predictions n_{pred}, the number of operators o, the recurrence degree d and the number of input terms l. In each plot, we use the following defaults for quantities which are not varied: $\tau = 10^{-10}$, $n_{pred} = 10$, $o \in [\![1, 10]\!]$, $d \in [\![1, 6]\!]$, $l \in [\![5, 30]\!]$.

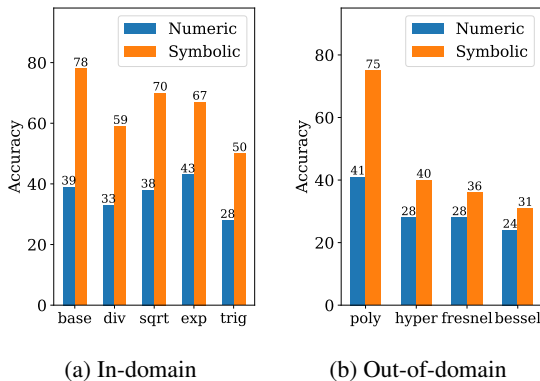(a) In-domain (b) Out-of-domain

Figure 2. **Accuracy of our models on various in-domain and out-of-domain groups.** We set $\tau = 10^{-10}$, $n_{pred} = 10$.

Section 2.1).

Unsurprisingly, accuracy degrades rapidly as the number of operators increases, particularly in the float setting where the operators are more diverse: the accuracy of the symbolic model drops from 100% for $o = 1$ to 10% for $o = 10$. We attempted a curriculum learning strategy to alleviate the drop, by giving higher probabilities to expressions with many operators as training advances, but this did not bring any improvement. Increasing the recurrence degree has a similar but more moderate effect: the accuracy decreases from 70% for $d = 0$ (non-recurrent expressions) to 20% for $d = 6$ in the float setup. Finally, we observe that shorter sequences are harder to predict as they give less information on the underlying recurrence; however, even when fed with less than 10 terms, our models achieve surprisingly high accuracies.

Ablation over operator families   To understand what kind of operators are the hardest for our float model, we bunch them into 5 groups:

- **base**: {add, sub, mul}.
- **division**: base + {div, inv}.
- **sqrt**: base + {sqrt}.
- **exponential**: base + {exp, log}.
- **trigonometric**: base + {sin, cos, tan, arcsin, arccos, arctan}.

Results are displayed in Figure 2a. We see that the main difficulties lie in division and trigonometric operators, but the performance of both models stays rather good in all categories.

Visualizing the embeddings   To give more intuition on the inner workings of our symbolic models, we display a t-SNE (van der Maaten & Hinton, 2008) projection of the embeddings of the integer model in Figure 3a and of exponent embeddings of the float model in Figure 3b.

Both reveal a sequential structure, with the embeddings organized in a clear order, as highlighted by the color scheme. In Appendix F, we study in detail the pairwise distances between embeddings, unveiling interesting features such as the fact that the integer model naturally learns a base-6 representation.

4. Out-of-domain generalization

In this section, we evaluate the ability of our model to generalize out-of-domain. Recurrence prediction being a previously unexplored branch of symbolic regression, there are no official benchmarks we can compare our models to. For integer sequences, we use a subset of OEIS as our out-of-domain benchmark; for float sequences, we use a generator with out-of-vocabulary constants and operators. In Appendix A, we also show that our models can and can
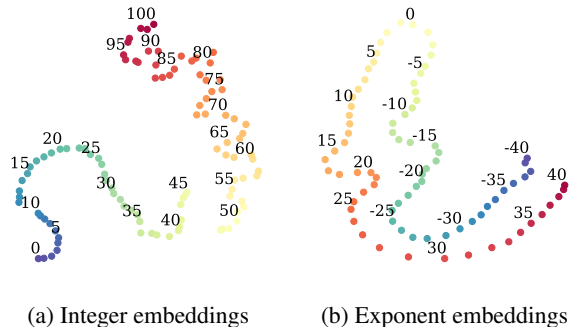
(a) Integer embeddings   (b) Exponent embeddings

Figure 3. **The number embeddings reveal intriguing mathematical structure**. We represented the t-SNE of the embeddings of the integer model and the exponent embeddings of the float model. We depicted the first 100 integer embeddings (10,000 in the model), and the exponent embeddings -40 to 40 (-100 to 100 in the model).

be made robust noise in the inputs.

4.1. Integer sequences: OEIS dataset

The Online Encyclopedia of Integer Sequences (OEIS) is an online database containing over 300,000 integer sequences. It is tempting to directly use OEIS as a testbed for prediction; however, many sequences in OEIS do not have a closed-form recurrence relation, such as the stops on the New York City Broadway line subway (A000053). These will naturally cause our model to fail.

Preprocessing Luckily, OEIS comes with keywords, and 22% of the sequences are labelled as "easy", meaning that there is a logic to find the next terms (although this logic is by no means easy in most cases). Note that this logic cannot always be formulated as a recurrence relation: for example, the sequence of primes or decimals of π are included in this category, but intractable for our models. We keep the first 10,000 of these sequences as our testbed. Evaluation consists in in showing our models the first $n_{input} \in \{15, 25\}$ terms of each sequence and asking it to predict the $n_{pred} \in \{1, 10\}$ following terms.

Results Results are reported in Table 7. With only $n_{input} = 15$ terms, the numeric model reaches an impressive accuracy of 53% at next term prediction, and 27% for predicting the next ten terms. The symbolic model achieves lower results, with 33% and 19% respectively; we attribute this to the large number of non-analytic sequences in the testbed. Nonetheless, this means that our model can retrieve a valid recurrence relation for almost a fifth of the sequences, which is rather impressive: we give a few interesting examples in Table 1. Increasing n_{input} to 25 increases our performances rather marginally.

As a comparison, we ran two built-in Mathematica func-

tions for the task at hand: FindSequenceFunction, which finds non-recurrent expressions, and FindLinearRecurrence, which finds linear recurrence relations. These functions are much more sensitive to the number of terms given as input: they obtain similar accuracies at $n_{input} = 15$, but FindLinearRecurrence performs significantly better at $n_{input} = 25$, while FindSequenceFunction performs pathologically worse. Both these functions perform less well than our symbolic model in all cases.

4.2. Float sequences: robustness to out-of-vocabulary tokens

One major difficulty in symbolic mathematics is dealing with out-of-vocabulary constants and operators: the model is forced to approximate them using its own vocabulary. We investigate these two scenarios separately for the float model.

Out-of-vocabulary constants The first possible source of out-of-vocabulary tokens are prefactors. For example, a formula as simple as $u_n = 0.33n$ is hard to predict perfectly, because our decoder only has access to integers between -10 and 10 and a few mathematical constants, and needs to write 0.33 as [div, add, mul, 3, 10, 3, mul, 10, 10]. To circumvent this issue, Biggio et al. (2021) use a separate optimizer to fit the prefactors, once the skeleton of the equation is predicted.

In contrast, our model is end-to-end, and is surprisingly good at approximating out-of-vocabulary prefactors with its own vocabulary. For $u_n = 0.33n$, one could expect the model to predict $u_n = n/3$, which would be a decent approximation. Yet, our model goes much further, and outputs $u_n = -\cos(3)n/3$, which is a better approximation. We give a few other spectacular examples in Table 2. Our model is remarkably capable of using the values of operators such as exp and arctan, as if it were able to perform computations internally.

To investigate the approximation capabilities of our model systematically, we evaluate the its performance when sampling the prefactors uniformly in $[-10, 10]$, rather than in $\{-10, -9, ...9, 10\} \cup \{e, \pi, \gamma\}$ as done usually. It is impossible for the symbolic model to perfectly represent the correct formulas, but since we are interested in its approximation capabilities, we set the tolerance to 0.01. Results are shown in Table 8. Unsurprisingly, the performance of the numeric model is unaffected as it does not suffer from any out-of-vocabulary issues, and becomes better than the symbolic model. However, the symbolic model maintains very decent performances, with its approximation accuracy dropping from 60% to 42%.

This approximation ability in itself an impressive feat, as the model was not explicitly trained to achieve it. It can poten-

Model	$n_{input} = 15$		$n_{input} = 25$	
	$n_{pred} = 1$	$n_{pred} = 10$	$n_{pred} = 1$	$n_{pred} = 10$
Symbolic (ours)	33.4	19.2	34.5	21.3
Numeric (ours)	53.1	27.4	54.9	29.5
FindSequenceFunction	17.1	12.0	8.1	7.2
FindLinearRecurrence	17.4	14.8	21.2	19.5

Table 7. **Accuracy of our integer models and Mathematica functions on OEIS sequences.** We use as input the first $n_{input} = \{15, 25\}$ first terms of OEIS sequences and ask each model to predict the next $n_{pred} = \{1, 10\}$ terms. We set the tolerance $\tau = 10^{-10}$.

Model	$[\![-10, 10]\!] \cup \{e, \pi, \gamma\}$		$\mathcal{U}(-10, 10)$	
	$n_{op} \leq 5$	$n_{op} \leq 10$	$n_{op} \leq 5$	$n_{op} \leq 10$
Symbolic	**81.9**	**60.7**	60.1	42.1
Numeric	72.4	60.4	**72.2**	**60.2**

Table 8. **Our symbolic model can approximate out-of-vocabulary prefactors.** We report the accuracies achieved when sampling the constants uniformly from $[\![-10, 10]\!] \cup \{e, \pi, \gamma\}$, as during training, versus sampling uniformly in $[-10, 10]$. We set $\tau = 0.01$ (note the higher tolerance threshold as we are considering approximation) and $n_{pred} = 10$.

tially have strong applications for mathematics, exploited by the recently proposed Ramanujan Machine (Raayoni et al., 2021): for example, if a sequence converges to a numerical value such as 1.64493, it can be useful to ask the model to approximate it, yielding $\pi^2/6$. In fact, one could further improve this approximation ability by training the model only on degree-0 sequences with constant leaves ; we leave this for future work.

Out-of-vocabulary functions   A similar difficulty arises when dealing with out-of-vocabulary operators, yet again, our model is able to express or approximate them with its own vocabulary. We show this by evaluating our model on various families of functions from `scipy.special`:

- **polynomials**: base + orthogonal polynomials of degree 1 to 5 (Legendre, Chebyshev, Jacobi, Laguerre, Hermite, Gegenbauer)
- **hyperbolic**: base + {sinh, cosh, tanh, arccosh, arcsinh, arctanh}
- **bessel**: base + {Bessel and modified Bessel of first and second kinds}
- **fresnel**: base + {erf, Faddeeva, Dawson and Fresnel integrals}.

The results in Figure 2b show that both the numeric and symbolic models cope surprisingly well with these functions. The symbolic model has more contrasted results than the numeric model, and excels particularly on functions which it can easily build with its own vocabulary such as polynomials. Surprisingly however, it also outperforms the numeric model on the other groups.

In Table 3, we show a few examples of the remarkable ability of our model to yield high-quality asymptotic approximations, either using a recurrence relation as for the Catalan numbers and the Dawson function, either with a non-recurrent expression as for the Bessel functions.

5. Conclusion

In this work, we have shown that Transformer models can successfully infer recurrence relations from observations. We applied our model to challenging out-of-distribution tasks, showing that it outperforms Mathematica functions on integer sequences and yields informative approximations of complex functions as well as numerical constants.

Scope of our approach   One may ask to what extent our model can be used for real-world applications, such as time-series forecasting. Although robustness to noise is an encouraging step in this direction, we believe our model is not directly adapted to such applications, for two reasons.

First, real-world time-series often cannot be described by a simple mathematical formula, in which case numeric approaches will generally outperform symbolic approaches. Second, even when they can be expressed by a formula, the latter will contain complex prefactors and non-deterministic terms which will make the task extremely challenging.

Future directions   To bring our model closer to real-world problems, we need our model to be able to handle arbitrary prefactors. We introduce a method to solve this problem in our follow-up work by introducing numeric tokens in the decoder (Kamienny et al., 2022).

Another important extension of our work is the setting of multi-dimensional sequences. With two dimensions, one could study complex sequences, which are a well-studied branch of mathematics given their relationship with fractals.

Finally, recurrence relations being a discrete version of differential equations, the most natural extension to this work is to infer differential equations from trajectories; this will be an important direction for future work.

References

Allamanis, M., Chanthirasegaran, P., Kohli, P., and Sutton, C. Learning continuous semantic representations of symbolic expressions. In *International Conference on Machine Learning*, pp. 80–88. PMLR, 2017.

Arabshahi, F., Singh, S., and Anandkumar, A. Towards solving differential equations through neural programming. In *ICML Workshop on Neural Abstract Machines and Program Induction (NAMPI)*, 2018.

Augusto, D. A. and Barbosa, H. J. Symbolic regression via genetic programming. In *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*, pp. 173–178. IEEE, 2000.

Biggio, L., Bendinelli, T., Neitz, A., Lucchi, A., and Parascandolo, G. Neural symbolic regression that scales. *arXiv preprint arXiv:2106.06427*, 2021.

Charton, F. Linear algebra with transformers. *arXiv preprint arXiv:2112.01898*, 2021.

Charton, F., Hayat, A., and Lample, G. Learning advanced mathematical computations from examples. *arXiv preprint arXiv:2006.06462*, 2020.

Cobbe, K., Kosaraju, V., Bavarian, M., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Davies, A., Velickovic, P., Buesing, L., Blackwell, S., Zheng, D., Tomasev, N., Tanburn, R., Battaglia, P., Blundell, C., Juhasz, A., et al. Advancing mathematics by guiding human intuition with ai. *Nature*, 2021.

Kaiser, Ł. and Sutskever, I. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Kamienny, P.-A., d'Ascoli, S., Lample, G., and Charton, F. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532*, 2022.

Kim, S., Lu, P. Y., Mukherjee, S., Gilbert, M., Jing, L., Čeperić, V., and Soljačić, M. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

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

McKay, B., Willis, M. J., and Barton, G. W. Using a tree structured genetic algorithm to perform symbolic regression. In *First international conference on genetic algorithms in engineering systems: innovations and applications*, pp. 487–492. IET, 1995.

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

Murari, A., Peluso, E., Gelfusa, M., Lupelli, I., Lungaroni, M., and Gaudio, P. Symbolic regression via genetic programming for data driven derivation of confinement scaling laws without any assumption on their mathematical form. *Plasma Physics and Controlled Fusion*, 57(1): 014008, 2014.

Nam, H., Kim, S., and Jung, K. Number sequence prediction problems for evaluating computational powers of neural networks, 2018.

Paccanaro, A. and Hinton, G. E. Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):232–244, 2001.

Petersen, B. K., Larma, M. L., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

Raayoni, G., Gottlieb, S., Manor, Y., Pisha, G., Harris, Y., Mendlovic, U., Haviv, D., Hadad, Y., and Kaminer, I. Generating conjectures on fundamental constants with the ramanujan machine. *Nature*, 590(7844):67–73, 2021.

Ragni, M. and Klein, A. Predicting numbers: An ai approach to solving number series. In Bach, J. and Edelkamp, S. (eds.), *KI 2011: Advances in Artificial Intelligence*, pp. 255–259. Springer Berlin Heidelberg, 2011.

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

Sahoo, S., Lampert, C., and Martius, G. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pp. 4442–4450. PMLR, 2018.

Saxton, D., Grefenstette, E., Hill, F., and Kohli, P. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

Schmidt, M. and Lipson, H. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

Sloane, N. J. The on-line encyclopedia of integer sequences. In *Towards mechanized mathematical assistants*, pp. 130–130. Springer, 2007.

Trask, A., Hill, F., Reed, S., Rae, J., Dyer, C., and Blunsom, P. Neural arithmetic logic units. *arXiv preprint arXiv:1808.00508*, 2018.

Valipour, M., You, B., Panju, M., and Ghodsi, A. Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*, 2021.

van der Maaten, L. and Hinton, G. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wu, C. W. Can machine learning identify interesting mathematics? an exploration using empirically observed laws. *arXiv preprint arXiv:1805.07431*, 2018.

A. Robustness to noise

One particularity of our model is that it is entirely trained and evaluated on synthetic data which is completely noise-free. Can our model also predict recurrence relations when the inputs are corrupted ? In this section, we show that the answer is yes, provided the model is trained with noisy inputs. For simplicity, we restrict ourselves here to the setup of float sequences, but the setup of integer sequences can be dealt with in a similar manner, trading continuous random variables for discrete ones.

Setup Considering the wide range of values that are observed in recurrent sequences, corruption via additive noise with constant variance, i.e. $u_n = f(n, \{u_i\}_{i<n}) + \xi_n, \xi_n \sim \mathcal{N}(0, \sigma)$ is a poor model of stochasticity. Indeed, the noise will become totally negligible when $u_n \gg 1$, and conversely, totally dominate when $u_n \ll 1$. To circumvent this, we scale the variance of the noise with the magnitude of the sequence, i.e. $\xi_n \sim \mathcal{N}(0, \sigma u_n)$, allowing to define a signal-to-noise ratio SNR $= 1/\sigma$. This can also be viewed as a multiplicative noise $u_n = f(n, \{u_i\}_{i<n})\xi, \xi \sim \mathcal{N}(1, \sigma)$.

Results To make our models robust to corruption in the sequences, we use stochastic training. This involves picking a maximal noise level σ_{train}, then for each input sequence encountered during training, sample $\sigma \sim \mathcal{U}(0, \sigma_{train})$, and corrupt the terms with a multiplicative noise of variance σ. At test time, we corrupt the input sequences with a noise of fixed variance σ_{test}, but remove the stochasticity for next term prediction, to check whether our model correctly inferred the deterministic part of the formula.

Results are presented in Table 9. We see that without the stochastic training, the accuracy of our model plummets from 43% to 1% as soon as noise is injected at test time. However, with stochastic training, we are able to keep decent performance even at very strong noise levels: at $\sigma_{test} = 0.5$, we are able to achieve an accuracy of 17%, which is remarkable given that the signal-to-noise ratio is only of two. However, this robustness comes at a cost: performance on the clean dataset is degraded, falling down to 30%.

$\sigma_{train/test}$	$\sigma_{test} = 0$	$\sigma_{test} = 0.1$	$\sigma_{test} = 0.5$
$\sigma_{train} = 0$	**43.3**	0.9	0.0
$\sigma_{train} = 0.1$	38.4	**31.9**	0.2
$\sigma_{train} = 0.5$	35.6	31.8	**11.1**

Table 9. **Our symbolic model can be made robust to noise in the inputs, with a moderate drop in performance on clean inputs.** We report the accuracy on expressions with up to 10 operators, for $n_{pred} = 10, \tau = 10^{-10}$, varying the noise level during training σ_{train} and evaluation σ_{test}.

B. Robustness to distribution shifts

Note that in all our training examples, the initial terms of the sequences are sampled in the range $[-10, 10]$. This naturally begs the question: can our model handle cases where the first terms lie outside this range?

Results, shown in Tab. 10, show that our symbolic models displays poor robustness to this distribution shift: their accuracy plummets by more than a factor of two. In comparison, the numeric model is significantly less affected.

Note that an easy fix to this issue would be to train with a larger range for the scale of the initial terms; nonetheless, this confirms the lack of out-of-distribution robustness of numeric-to-symbolic models, previously demonstrated by (Charton, 2021).

Model	Integer		Float	
	$[\![-10, 10]\!]$	$[\![-100, 100]\!]$	$[-10, 10]$	$[-100, 100]$
Symbolic	80.3	30.5	60.7	27.9
Numeric	75.5	67.4	60.4	60.2

Table 10. **The symbolic models suffer from distribution shifts.** We report the in-domain accuracies obtained when sampling the first terms of the sequences uniformly in $[\![-100, 100]\!]$ (integer) and $[-100, 100]$ (float) instead of $[\![-10, 10]\!]$ and $[-10, 10]$ seen during training. We set $\tau = 0.01$ and $n_{pred} = 10$.

C. The effect of expression simplification

One issue with symbolic regression is the fact that a mathematical expression such as `mul, 2, cos, n` can be written in many different ways. Hence, cross-entropy supervision to the tokens of the expression can potentially penalize the model for generating the same formula written in a different way (e.g. `mul, cos, n, 2` or `mul, cos, n, add, 1, 1`). To circumvent this issue, (Petersen et al., 2019) first predict the formula, then evaluate it and supervise the evaluations to those of the target function. Yet, since the evaluation step is non-differentiable, they are forced to use a Reinforcement Learning loop to provide reward signals. In our framework, we noticed that such an approach is actually unnecessary.

Instead, one could simply preprocess the mathematical formula to simplify it with SymPy (Meurer et al., 2017) before feeding it to the model. This not only simplifies redundant parts such as `add, 1, 1 → 2`, but also gets rid of the permutation invariance `mul, x, 2 = mul, 2, x` by following deterministic rules for the order of the expressions. However, and rather surprisingly, we noticed that this simplification does not bring any benefit to the predictive power of our model : although it lowers the training loss (by getting rid of permutation invariance, it lowers cross-entropy), it does not improve the test accuracy, as shown in Fig. 4. This suggests that expression syntax is not an issue for our model: the hard part of the problem indeed lies in the mathematics.

Aside from predictive power, SymPy comes with several advantages and drawbacks. On the plus side, it enables the generated expressions to be written in a cleaner way, and improves the diversity of the beam. On the negative side, it slows down training, both because it is slow to parse complex expressions, and because it often lengthens expression since it does not handle division (SymPy rewrites `div,a,b` as `mul,a,pow,b,-1`). Additionally, simplification actually turns out to be detrimental to the out-of-domain generalization of the float model. Indeed, to generate approximations of out-of-vocabulary prefactors, the latter benefits from non-simplified numerical expressions. Hence, we chose to not use SymPy in our experiments.
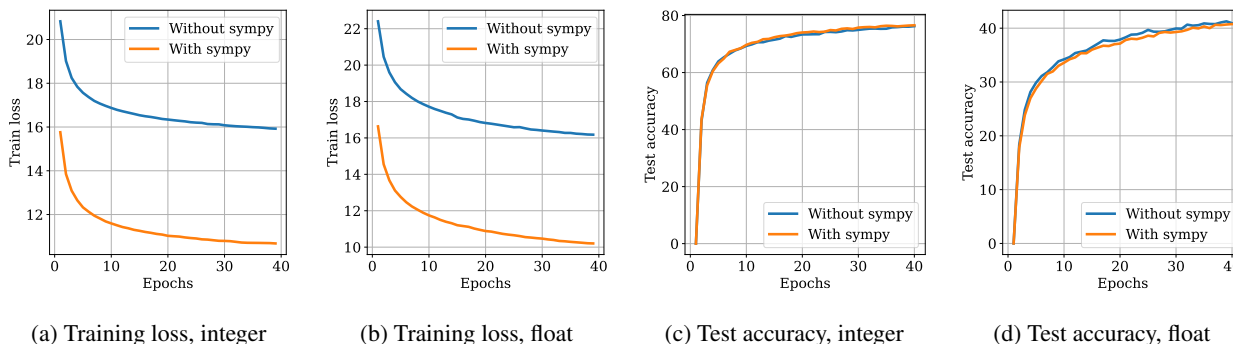

(a) Training loss, integer   (b) Training loss, float   (c) Test accuracy, integer   (d) Test accuracy, float

Figure 4. **Simplification reduces the training loss, but does not bring any improvement in test accuracy**. We displayed the first 40 epochs of training of our symbolic models.

D. Does memorization occur?

It is natural to ask the following question: due to the large amount of data seen during training, is our model simply memorizing the training set ? Answering this question involves computing the number of possible inputs sequences N_{seq} which can be generated. To estimate this number, calculating the number of possible mathematical expressions N_{expr} is insufficient, since a given expression can give very different sequences depending on the random sampling of the initial terms. Hence, one can expect that N_{expr} is only a very loose lower bound for N_{seq}.

Nonetheless, we provide the lower bound N_{expr} as a function of the number of nodes in Fig. 5, using the equations provided in (Lample & Charton, 2019). For small expressions (up to four operators), the number of possible expressions is lower or similar to than the number of expressions encountered during training, hence one cannot exclude the possibility that some expressions were seen several times during training, but with different realizations due to the initial conditions. However, for larger expressions, the number of possibilities is much larger, and one can safely assume that the expressions encountered at test time have not been seen during training.
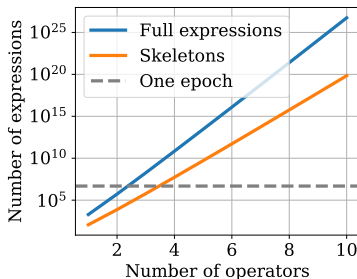
Figure 5. **Our models only see a small fraction of the possible expressions during training.** We report the number of possible expressions for each number of operators (skeleton refers to an expression with the choice of leaves factored out). Even after a hundred epochs, our models have only seen a fraction of the possible expressions with more than 4 operators.

E. The issue of finite precision

As explained in the main text, our model tends to ignore subdominant terms in expressions with terms of vastly different magnitudes, partly due to finite precision. For example, when using a float precision of $p = 4$ digits, we obtain a discretization error of 10^5 for numbers of magnitude 10^9. However, there exists two methods to circumvent this issue.

Increasing the precision   Naively increase the precision p would cause the vocabulary size of the encoder to rapidly explode, as it scales as 10^p. However, one can instead encode the mantissa on multiple tokens, as performed for integers. For example, using two tokens instead of one, e.g. encoding π as `+ 3141, 5926, E-11`, doubles the precision while increasing the sequence length only by 33%.

This approach works well for recurrence prediction, but slightly hampers the ability of the model to approximate prefactors as shown in Tab. 2, hence we did not use it in the runs presented in this paper.

Iterative refinement   Another method to improve the precision of the model is to use an iterative refinement of the predicted expression, akin to perturbation theory in physics.

Consider, for example, the polynomial $f(x) = \sum_{k=0}^{d} a_k x^k$, for which our model generally predicts $\hat{f}(x) = \sum_{k=0}^{d} \hat{a}_k x^k$, with the first coefficient correct ($\hat{a}_d = a_d$) but potentially the next coefficients incorrect ($\hat{a}_k \neq a_k$ for $k < d$). One can correct these subdominant coefficients iteratively, order by order. To obtain the term a_{d-1}, fit the values of $g(x) = f(x) - \hat{f}(x) = \sum_{k=1}^{d-1}(a_k - \hat{a}_k)$. Then fit the values of $h(x) = g(x) - \hat{g}(x)$, etc. By iterating this procedure k times, one can obtain the k highest coefficients a_k.

We checked that this method allows us to approximate any polynomial function. One could in fact use iterative refinement to predict the Taylor approximation of any function, or use a similar approach to catch multiplicative corrections, by fitting $g(x) = f(x)/\hat{f}(x)$; we leave these investigations for future work.

F. Structure of the embeddings

To gain better understanding on the number embeddings of our models, we depict similarity matrices whose element (i, j) is the cosine similarity between embeddings i and j in Fig. 6.

Integer sequences   The numeric and symbolic similarity matrices look rather similar, with a bright region appearing around the line $y = x$, reflecting the sequential nature of the embeddings. In both cases, we see diagonal lines appear : these correspond to lines of common divisors between integers. Strikingly, these lines appear most clearly along multiples of 6 and 12, especially in the symbolic model, suggesting that 6 is a natural base for reasoning. These results are reminiscent of the much earlier explorations of Paccanaro & Hinton (2001).

Float sequences   Both for the numeric and symbolic setups, the brightest regions appear along the diagonal lines $y = x$ and $y = -x$, reflecting respectively the sequential nature of the embeddings and their symmetry around 0. The darkest
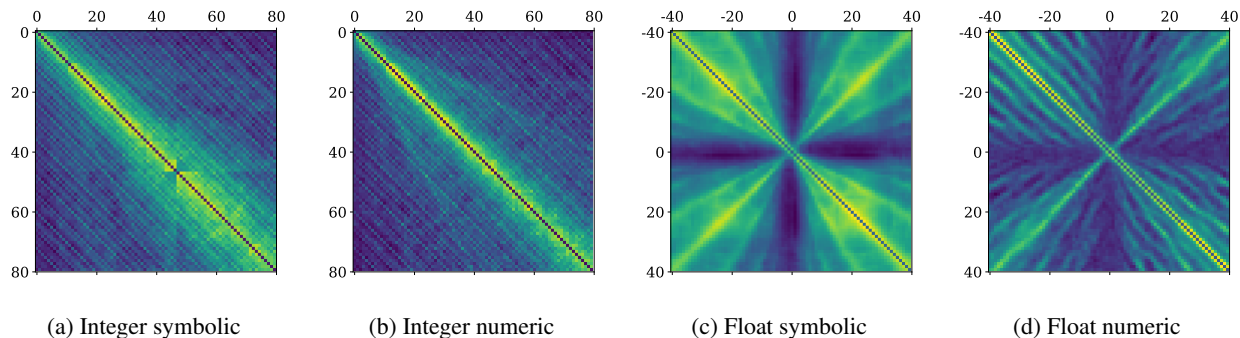
(a) Integer symbolic   (b) Integer numeric   (c) Float symbolic   (d) Float numeric

Figure 6. **The similarity matrices reveal more details on the structure of the embeddings**. The element (i, j) is the cosine similarity between embeddings i and j.

regions appears around the vertical line $x = 0$ and the horizontal line $y = 0$, corresponding to exponents close to zero: these exponents strongly overlap with each other, but weakly overlap with the rest of the exponents. Interestingly, dimmer lines appear in both setups, but follow a very different structure. In the symbolic setup, the lines appear along lines $y = \pm x/k$, reminiscent of the effect of polynomials of degree k. In the numeric setup, the lines are more numerous, all diagonal but offset vertically.

G. Visualizations

Success and failure modes In Fig. 7, we show a few examples of success and failure modes of our symbolic models. The failure modes are particularly interesting, as they reflect the strong behavioral difference between our symbolic models and models usually used for regression tasks.

The latter generally try to interpolate the values of the function they are given, whereas our symbolic model tries to predict the expression of the function. Hence, our model cannot simply "overfit" the inputs. A striking consequence of this is that in case of failure, the predicted expression is wrong both on the input points (green area) and the extrapolation points (blue area).

In some cases, the incorrectly predicted formula provides a decent approximation of the true function (e.g. when the model gets a prefactor wrong). In others, the predicted formula is catastrophically wrong (e.g. when the model makes a mistake on an operator or a leaf).

Training curves In Fig. 8, we show the training curves of our models, presenting an ablation over the tolerance τ, the number of predictions n_{pred}, the number of operators o, the recurrence degree d and the number of input points l, as explained in the main text.

Attention maps In Fig. 9, we provide attention maps for the 8 attention heads and 4 layers of our Transformer encoders. Clearly, different heads play very different roles, some focusing on local interactions and others on long-range interactions. However, the role of different layers is hard to interpret.
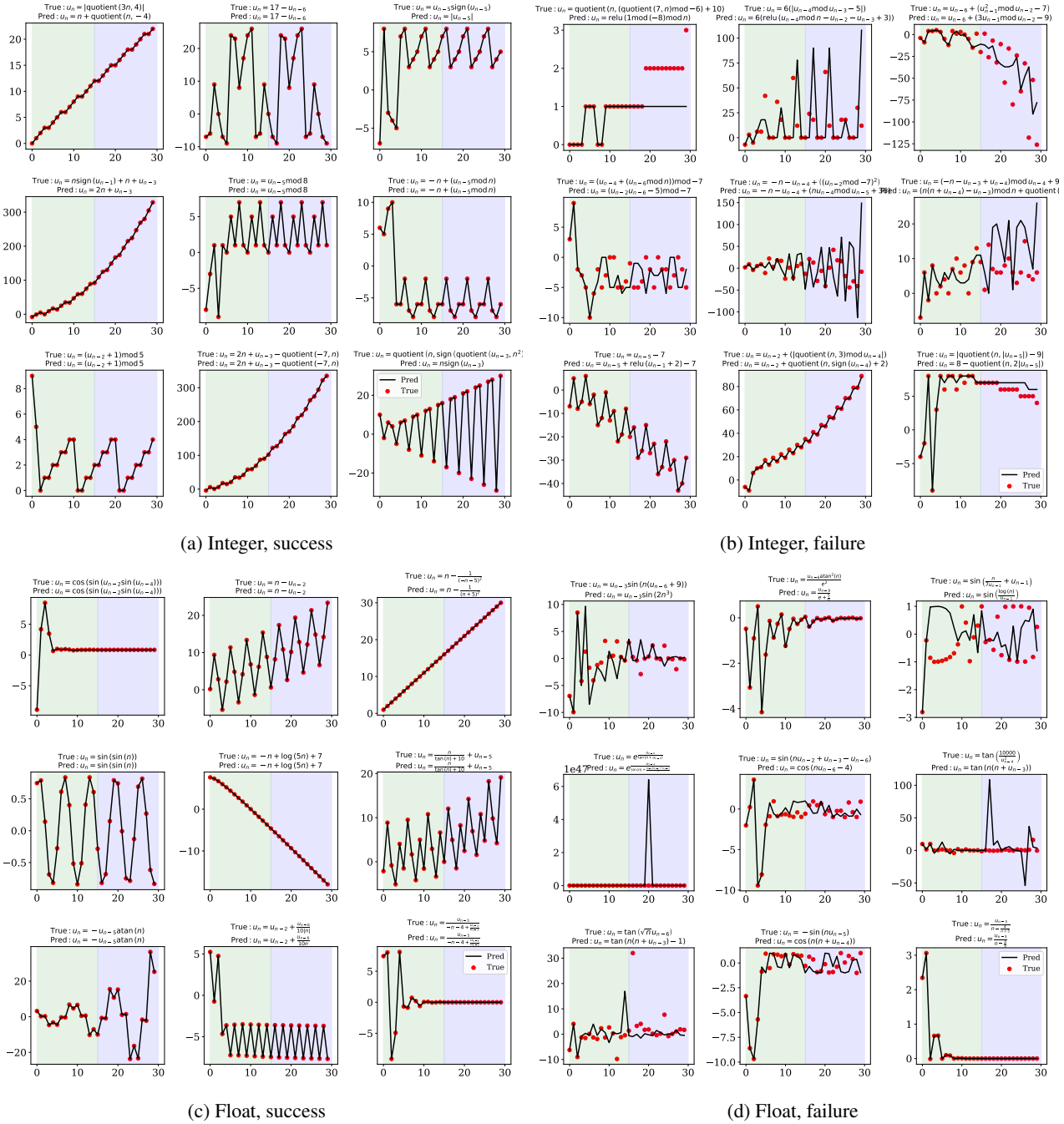
(a) Integer, success

(b) Integer, failure

(c) Float, success

(d) Float, failure

Figure 7. **Success and failure modes of our models.** The models are fed the first 15 terms of the sequence (green area) and predict the next 15 terms (blue area). We randomly selected expressions with 4 operators from our generator, and picked the first successes and failures.

(a) Integer symbolic

(b) Integer numeric

(c) Float symbolic

(d) Float numeric
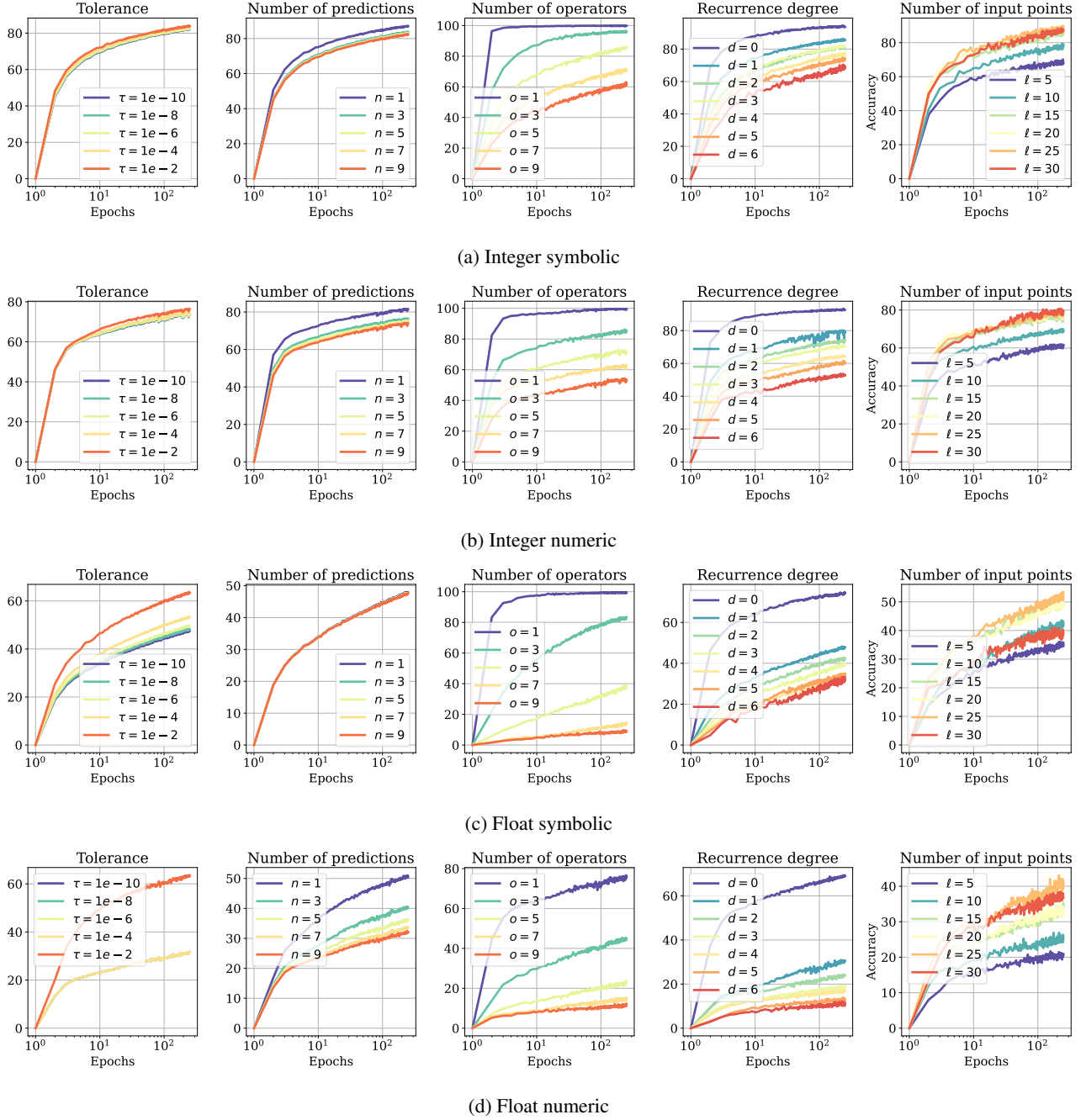
Figure 8. **Training curves of our models.** We plot the accuracy of our models at every epoch, evaluated of 10,000 sequences generated from the same distribution as during trianing. From left to right, we vary the tolerance τ, the number of predictions n_{pred}, the number of operators o, the recurrence degree d and the number of input terms l. In each plot, we use the following defaults for quantities which are not varied: $\tau = 10^{-10}$, $n_{pred} = 10$, $o \in [\![1, 10]\!]$, $d \in [\![1, 6]\!]$, $l \in [\![5, 30]\!]$.
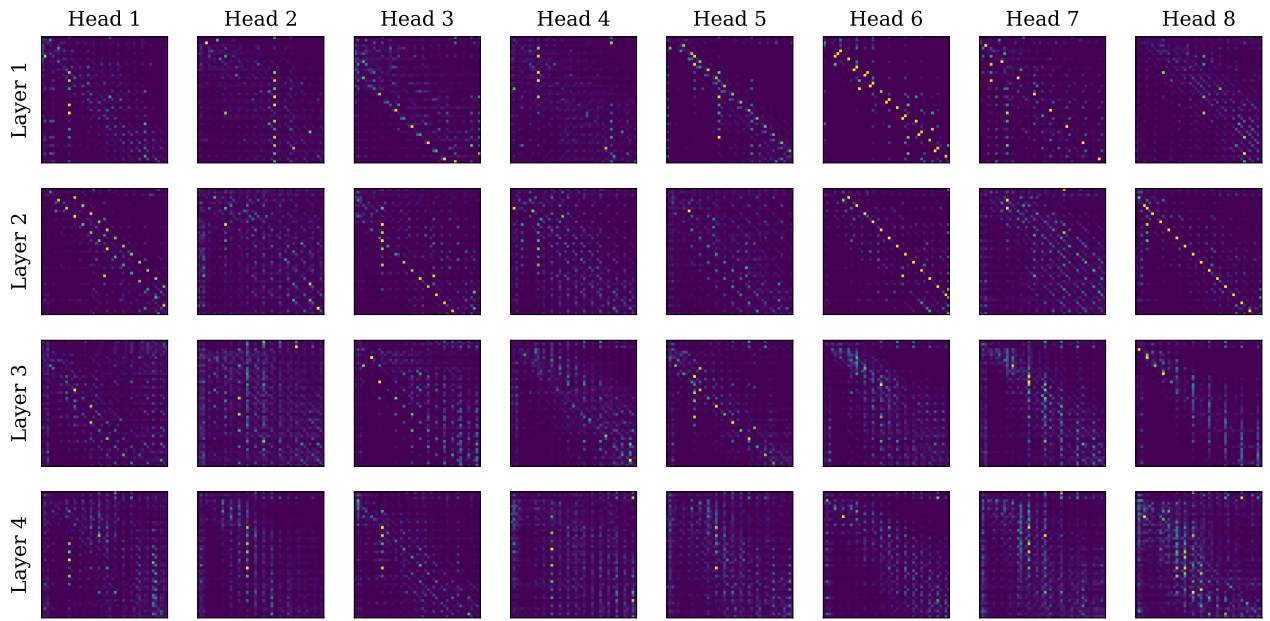
(a) Integer
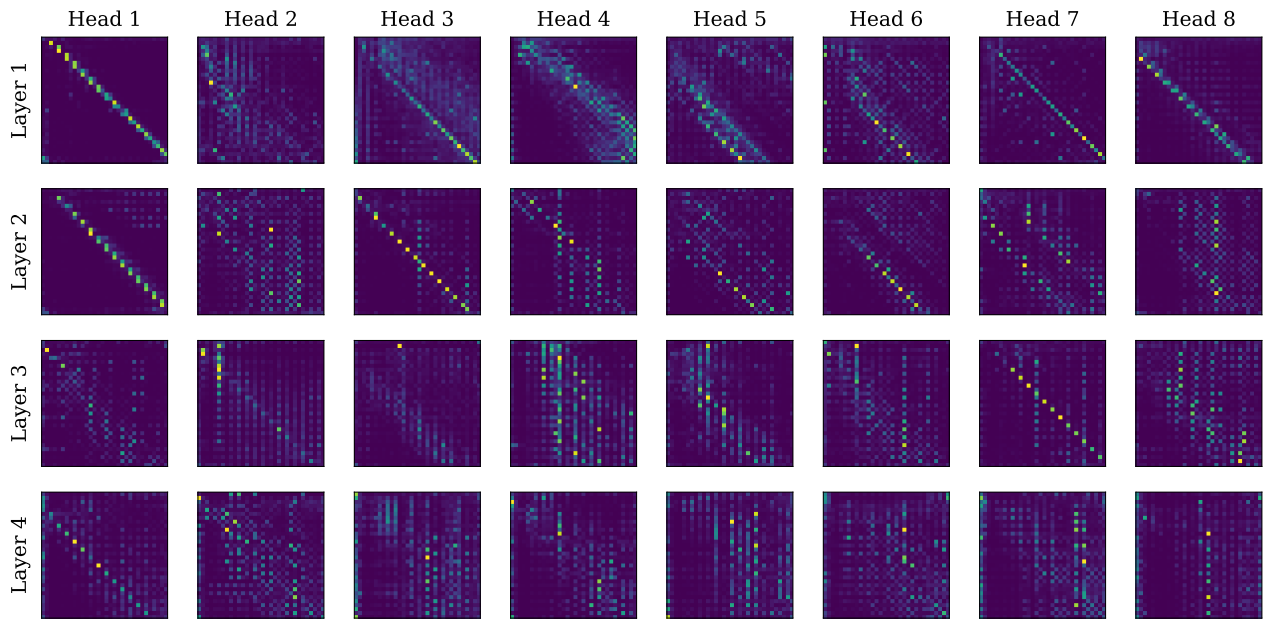
(b) Float

Figure 9. **Attention maps of our integer model and float models**. We evaluated the integer model on the first 25 terms of the sequence $u_n = -(6 + u_{n-2}) \bmod n$ and the float model on the first 25 terms of the sequence $u_n = \exp(\cos(u_{n-2}))$.