Symbolic Brittleness in Sequence Models:
on Systematic Generalization in Symbolic Mathematics

Sean Welleck,[1,2] **Peter West,**[1] **Jize Cao,**[1] **Yejin Choi**[1,2]

[1] Paul G. Allen School of Computer Science & Engineering, University of Washington
[2] Allen Institute for Artificial Intelligence
wellecks@uw.edu

Abstract

Neural sequence models trained with maximum likelihood estimation have led to breakthroughs in many tasks, where success is defined by the gap between training and test performance. However, their ability to achieve stronger forms of generalization remains unclear. We consider the problem of symbolic mathematical integration, as it requires generalizing systematically beyond the test set. We develop a methodology for evaluating generalization that takes advantage of the problem domain's structure and access to a verifier. Despite promising in-distribution performance of sequence-to-sequence models in this domain, we demonstrate challenges in achieving robustness, compositionality, and out-of-distribution generalization, through both carefully constructed manual test suites and a genetic algorithm that automatically finds large collections of failures in a controllable manner. Our investigation highlights the difficulty of generalizing well with the predominant modeling and learning approach, and the importance of evaluating beyond the test set, across different aspects of generalization.[1]

1 Introduction

Despite their success, recent studies reveal undesirable properties of conventional neural sequence models, such as assigning high-probabilities to unrealistic sequences (Holtzman et al. 2020; Welleck et al. 2020), susceptibility to adversarial attacks (Wallace et al. 2019), and limited generalization on symbolic tasks (Saxton et al. 2019; Nogueira, Jiang, and Li 2021), even with very large models and datasets (Henighan et al. 2020). Despite these drawbacks, Lample and Charton (2019) recently demonstrated that a standard sequence-to-sequence model, which we call a *neural sequence integrator*, performs surprisingly well at *symbolic integration*, solving problems that are beyond traditional symbolic solvers and achieving near perfect test accuracy.

Recent studies suggest that achieving *strong and systematic generalization* is difficult with vanilla sequence-to-sequence methods, as they latch onto regularities in the training data, learning dataset-specific solutions that do not generalize beyond the training distribution (e.g. Agrawal, Batra, and Parikh (2016); Lake and Baroni (2018); Bahdanau et al. (2019); Hupkes et al. (2020)). Symbolic integration

---

[1]Code will be made available at:
https://github.com/wellecks/symbolic_generalization

Input	Integral	Prediction	
$30\cos(39x)$	$\frac{10}{13}\sin(39x)$	$\frac{10}{13}\sin(39x)$	✓
$17\cos(83x)$	$\frac{17}{83}\sin(83x)$	$\frac{1}{17}\sin(83x)$	✗
$34\cos(77x)$	$\frac{34}{77}\sin(77x)$	$\sin(77x)$	✗
x^{209}	$\frac{1}{210}x^{210}$	$\frac{1}{210}x^{210}$	✓
x^{764}	$\frac{1}{765}x^{765}$	$\frac{1}{765}x^{765}$	✓
$x^{209}+x^{764}$	$\frac{1}{210}x^{210}+\frac{1}{765}x^{765}$	$\frac{1}{205}x^{205}$	✗
-241	$-241x$	$-239x-14400$	✗
123^x	$\frac{123^x}{\log(123)}$	$\frac{123^x}{1+\log(123)}$	✗
$4^x+x^{465}+1$	$\frac{x^{466}}{466}+x+\frac{4^x}{\log(4)}$	$\frac{x^{466}}{466}+x+e^x$	✗

Table 1: Despite its impressive ability to integrate equations that are out of reach for traditional symbolic solvers, the neural sequence integrator shows deficiences in **robustness** (top) and **compositionality** (middle), and fails on **adversarial** problems discovered by SAGGA (bottom).

– finding the integral of a mathematical function – specifically requires these forms of generalization, as it involves an underlying structure that extends beyond this fixed training distribution. For instance, the rule $\int k = kx + C$ applies to all constants k, and the sum rule $\int f_1 + \int f_2 = \int(f_1 + f_2)$ means that integrating two functions correctly should imply integrating their sum correctly. Symbolic integration also offers a structured problem domain and a verifier for evaluating whether a proposed solution is correct, making it an effective setting for evaluating generalization. As the neural sequence integrator relies on a common recipe– a large-scale transformer trained to maximize the likelihood of a training set of input-output sequences – it is especially interesting to study whether it generalizes systematically.

In this paper, we find a discrepancy between the traditional notion of generalization captured by test set accuracy and the generalization needed in symbolic mathematics. While the model's test accuracy is nearly perfect, we find this breaks down when testing its *robustness*, *compositionality*, and *out-of-distribution generalization* (e.g. Table 1). We describe a methodology for evaluating these aspects, by constructing problem sets and developing a genetic
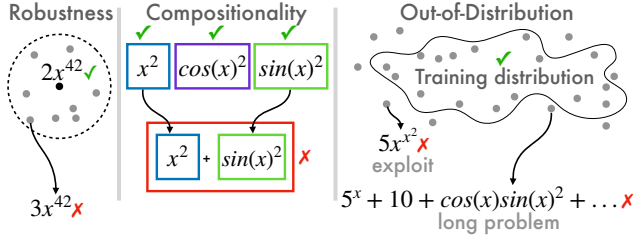
Figure 1: Illustrating *robustness*, *compositionality*, and *out-of-distribution* deficiencies in the neural sequence integrator.

algorithm, SAGGA (Symbolic Archive Generator with Genetic Algorithms), that automatically discovers diverse and targeted failures. We find that successfully integrating an in-distribution problem does not imply success on nearby problems, despite being governed by the same underlying rule (*robustness*). Moreover, the model often succeeds on a collection of problems without being able to systematically compose those problems (*compositionality*), and struggles to generalize to longer problems, larger values, and functions not covered in training (*out-of-distribution*).

In addition to the model's approximate mode being incorrect – i.e. the most probable sequence returned by beam search – the deficiencies are present deeper into its ranked list of candidate solutions, impacting the model's effectiveness in a *search-and-verify* setting. Overall, our investigation highlights the difficulty of achieving robustness, compositionality, and out-of-distribution generalization with the predominant modeling and learning approach, and the importance of evaluating beyond the test set, across aspects of generalization that are required by the task at hand.

2 Problem Setup

Symbolic integration is the problem of finding the integral $\mathbf{y}$ of an input equation $\mathbf{x}$. For instance, $x^2/2$ is the integral of x, up to an additive constant.

Neural sequence integrator. Lample and Charton (2019) frame symbolic integration as a sequence-to-sequence problem. In this view, input and output equations $\mathbf{x}$ and $\mathbf{y}$ are prefix-notation sequences. The *neural sequence integrator* uses a 6-layer transformer (Vaswani et al. 2017) to model the distribution $p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T_\mathbf{y}} p_\theta(y_t|y_{<t}, \mathbf{x})$ by training the model to maximize the log-likelihood of a set of training problems, $\arg\max_\theta \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \log p_\theta(\mathbf{y}|\mathbf{x})$. Given a trained model and input $\mathbf{x}$, a set of predicted solutions ranked by a model score is obtained by beam search, denoted $\{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_k\} = f_\theta(\mathbf{x}; k, b)$, where b is beam size and k is the number of candidates saved for evaluation. For brevity we omit b in the discussion unless necessary.

Evaluation. The standard practice is to evaluate a candidate $\hat{\mathbf{y}}$ by checking whether the derivative of $\hat{\mathbf{y}}$ is equivalent to $\mathbf{x}$ using a symbolic solver (e.g. Sympy). In the *maximum-a-posteriori* (MAP) setting, the model's output is considered correct if its *top-ranked* candidate $\hat{\mathbf{y}}_1$ is correct. This criterion is relaxed in the *search-and-verify* setting, where the

model's output is considered correct if *any* of its k candidates $\{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_k\}$ is correct. In this view, the neural network narrows the search space to a small set of candidates that are checked, trading off correctness for search and verification cost. We denote checking k candidate solutions as,

$$m(\mathbf{x}, f_\theta(\mathbf{x}; k)) = \begin{cases} 0 & \mathbf{x} \equiv \frac{d}{dx}\hat{\mathbf{y}}_i \text{ for any } i \in 1 \text{ to } k, \\ 1 & \text{otherwise.} \end{cases} \quad (1)$$

In other words, $m(\cdot, \cdot)$ is 1 when the model *fails* to predict the correct integral, and 0 when the model *succeeds*. We measure the proportion of failures on problems $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ using k candidate solutions per problem as:

$$\texttt{Fail@k}(f_\theta, X) = \frac{1}{N} \sum_{\mathbf{x}\in X} m(\mathbf{x}, f_\theta(\mathbf{x}; k)). \quad (2)$$

`Fail@k` is 0 when the model correctly integrates all of the problems in X, and increases towards 1 as it fails to integrate more problems. Evaluating a model's performance in the MAP setting corresponds to evaluating `Fail@1`, while the search-and-verify setting with a budget of $k > 1$ candidates uses `Fail@k`. We omit k in $f_\theta(\mathbf{x}; k)$ unless necessary.

2.1 Experiment Structure

We structure our investigation into three parts (Figure 1). We begin close to the model's training distribution, evaluating *robustness* to small perturbations of in-distribution problems and simple functions. We then ask whether learning to integrate a collection of functions implies that the model can integrate a *composition* of those functions. Finally we depart from the training distribution by studying *extrapolation* to larger problems and values, then by finding adversarial *exploits* that expose gaps in the training distribution.

Experimental setup. We use the implementation and pre-trained model from Lample and Charton (2019) for all of our experiments, specifically the `FWD+BWD+IBP` model which obtained top-10 accuracies of 95.6%, 99.5%, and 99.6% on their publicly available test sets.[2] Our evaluation is based on their code, we use their utilities for inputs and outputs, and by default use beam search with beam-size 10. Following the authors, we use Sympy to check whether the derivative of a prediction is equal to the original problem. We generously count the prediction as correct if a timeout occurs. See the Apppendix for additional details.

2.2 Automatic Problem Discovery with SAGGA

Automatically finding problems that expose deficiencies requires a non-differentiable cost (Equation 1), satisfying constraints for valid equations, and finding diverse problem sets to characterize each aspect of generalization. To address these challenges, we develop SAGGA (**S**ymbolic **A**rchive **G**eneration with **G**enetic **A**lgorithms), a gradient-free genetic algorithm which iteratively finds diverse failures.

At each iteration, SAGGA mutates a seed set of problems by modifying each problem's equation tree, ensuring that the

---

[2]https://github.com/facebookresearch/SymbolicMathematics/, commit `4596d07`.

Algorithm 1: SAGGA. Each seed problem denoted as $\hat{\mathbf{x}}$, mutated problem as $\tilde{\mathbf{x}}$, archived problem as $\mathbf{x}$.

Parameters: Fitness $F(f_\theta, \mathbf{x}) \to \mathbb{R}$, `mutate` and `seed` strategies, archive size N.
Output: Problem archive $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$.
$\mathcal{D} = \emptyset$ `// initial archive`
$\hat{\mathbf{x}}_{1:M}^{(0)} = \text{seed}(\mathcal{D}, \emptyset)$ `// initial seed`
while $|\mathcal{D}| < N$ **do**
    `// generate mutations`
    $\tilde{\mathbf{x}}_{1:M'}^{(i)} = \text{mutate}(\hat{\mathbf{x}}_{1:M}^{(i)})$
    `// select problems by fitness`
    $\mathbf{x}_{1:M''}^{(i)} = \text{select}(F, \tilde{\mathbf{x}}_{1:M'}^{(i)})$
    `// archive selected problems`
    $\mathcal{D} = \mathcal{D} \cup \mathbf{x}_{1:M''}^{(i)}$
    `// choose next seed`
    $\hat{\mathbf{x}}_{1:M}^{(i+1)} = \text{seed}(\mathcal{D}, F, \tilde{\mathbf{x}}_{1:M'}^{(i)})$

---

resulting candidates are valid equations. The candidates are scored by a fitness function – i.e. according to whether the neural sequence integrator fails to integrate the problem and other desired constraints – and the highest-fitness candidates are saved in a problem archive. The next seed set is then formed to balance diversity and fitness, by clustering candidates and selecting the highest-fitness members of each cluster. SAGGA continues until the archive contains a target number of problems. Algorithm 1 summarizes SAGGA.

SAGGA offers control over the types of problems that it discovers through its seed problems, fitness function, and mutation strategy. We detail our choices for each kind of generalization in their respective sections, and show default settings and further implementation details in the Appendix.

3 Robust or Brittle?

First, we study whether the model's strong test-set performance adequately represents its *robustness*. Robustness tells us whether the integration model systematically solves all problems in a neighborhood governed by a generalizable pattern; for instance a model that solves $\int 26x^{42}$ should solve $\int 53x^{42}$. We study problems that are nearby to those from the original test distribution, as well as to simple primitive functions that offer fine-grained, interpretable control.

A **robust** model is stable to small perturbations in input, meaning that it gets nearby problems $\tilde{\mathbf{x}}$ correct when it gets a problem $\mathbf{x}$ correct. Formally, let $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ contain problems that the model gets correct, $\sum_{\mathbf{x} \in X} m(\mathbf{x}, f_\theta(\mathbf{x})) = 0$, and let $\mathcal{N}_d(\mathbf{x})$ be a set of problems that are nearby to $\mathbf{x}$ according to a distance $d(\mathbf{x}, \tilde{\mathbf{x}})$. We measure robustness by measuring failures on nearby problems,

$$\text{Fail@k}(f_\theta, X_\mathcal{N}), \quad (3)$$

where $X_\mathcal{N} = \bigcup_{\mathbf{x} \in X} \mathcal{N}_d(\mathbf{x})$. We measure this quantity by varying (i) the neighborhood $\mathcal{N}_d(\mathbf{x})$ used to generate nearby problems, and (ii) the seed problems X to consider. Below, we will refer to a problem as $\mathbf{x}$ or f interchangeably.

Type	Test	Fail@50	Fail@10	Fail@1
coeff	$k_1 \ln(k_2 x)$	0.0	0.0	0.0
	$k_1 x$	0.0	0.0	0.0
	$k_1 x^{42}$	0.0	6.1	45.5
	$k_1 \exp(k_2 x)$	15.4	20.8	30.3
	$k_1 \sin(k_2 x)$	6.6	19.6	29.7
	$k_1 \cos(k_2 x)$	10.6	20.7	28.2
	$k_1 \tan(k_2 x)$	13.9	17.4	24.2
coeff	$1/k \cdot f$	5.9	12.0	13.7
coeff	$k \cdot f$	5.4	5.8	16.3
+exp	$f + e^x$	0.9	1.6	3.3
+ln	$f + \ln(x)$	1.9	3.2	5.3

Table 2: *Robustness* results with **simple primitives** (top) and **validation problems** f which the model correctly integrates (bottom). Coefficients are sampled from [1, 100].

Input	Integral	Prediction	
$30\cos(39x)$	$\frac{10}{13}\sin(39x)$	$\frac{10}{13}\sin(39x)$	✓
$17\cos(83x)$	$\frac{17}{83}\sin(83x)$	$\frac{1}{17}\sin(83x)$	✗
$34\cos(77x)$	$\frac{34}{77}\sin(77x)$	$\sin(77x)$	✗
$26x^{42}$	$\frac{26}{43}x^{43}$	$\frac{26}{43}x^{43}$	✓
$88x^{42}$	$\frac{88}{43}x^{43}$	$8x^{43}$	✗
$53x^{42}$	$\frac{53}{43}x^{43}$	$(x^{44}+x)/x$	✗

Table 3: *Robustness* examples. We show the model's top prediction (beam search, size 10). Note that $(x^{44}+x)/x = x^{43}+1$; its derivative is $43x^{42}$ and is hence incorrect.

3.1 Manually Testing Robustness

To define nearby problems, we first consider manual templates which minimally perturb a problem f, e.g.

$$k \cdot f, \quad f + \ln x, \quad \dots$$

These problems are nearby f in the sense that a single operation is added to the problem's equation tree, or a small number of node values are changed in the tree.

Brittleness on simple primitive functions. We first investigate whether the neural sequence integrator is robust on *simple primitive functions*, since they make up more complicated functions and are frequently entered by real-world users. We use a manual neighborhood which yields,

$$X_\mathcal{N} = \{k_1 \ln(k_2 x), \quad k_1 \exp(k_2 x), \quad k_1 x, \quad k_1 x^{42},$$
$$k_1 \sin(k_2 x), \quad k_1 \cos(k_2 x), \quad k_1 \tan(k_2 x)\},$$

where $k_1 \sim \mathcal{U}(a, b)$ and $k_2 \sim \mathcal{U}(a, b)$ are randomly sampled coefficients from a range (a, b). We use $[0, 100]$ which is covered by the training distribution and evaluate on 1,000 k_1, k_2 pairs sampled without replacement for each primitive.

Table 2 shows the results. On a positive note, the neural sequence integrator is robust on the primitives $k_1 x$ and $k_1 \ln(k_2 x)$. The integral of $k_1 x$ is $\frac{k_1}{2} x^2$, so the model learned to divide by 2 for these cases. The integral of $\ln$ involves copying the coefficients into a correct template (that

is, $\int k_1 \ln(k_2 x) = k_1 x(\ln(k_2 x) - 1))$, and the neural sequence integrator learned this behavior.

On the other hand, the model is surprisingly brittle on the other primitives. These require dividing coefficients (e.g. $\int k_1 \cos(k_2 x) = \frac{k_1}{k_2} \sin(k_2 x)$). The failure rate shows that the model has not perfectly learned the required division behavior. Moreover, despite learning a 'division by 2' rule for integrating $k_1 x$, the neural sequence integrator's failures on $k_1 x^{42}$ indicate that it did not perfectly learn an analogous 'division by 43' rule. Table 3 shows examples.

Test accuracy does not imply robustness. Next, we want to see whether the neural sequence integrator's strong test accuracy implies that it is robust on test problems. We use the validation set, and perturb *validation problems that the model correctly integrates* using the neighborhoods,

$$X_{\mathcal{N}_1} = \{\frac{1}{k} f, \quad k \cdot f\}, \quad X_{\mathcal{N}_2} = \{f + e^x, \quad f + \ln(x)\},$$

where $k \sim \mathcal{U}(1, 100)$. The first set multiplies the function by a constant, while the second adds a single primitive.

Table 2 shows the results. Despite achieving perfect accuracy on the original problems, the model frequently fails under the slight perturbations. The local neighborhood around validation examples reveals deficiencies in robustness that are not evident from validation performance alone, aligning with findings in NLP tasks (Gardner et al. 2020).

3.2 Automatically Finding Robustness Failures

Next, we use SAGGA to automatically discover robustness failures in the neighborhood of a seed set of problems.

Discovering brittleness near simple problems. First, we run SAGGA and only allow it to mutate leaves in a problem's equation tree into a random integer. The problems are nearby in the sense that the tree's structure is not changing, only a small number of its leaf values. We use SAGGA to mutate the leaves of seed sets of 9 polynomials $X_{\texttt{poly}}$ and 9 trigonometric functions $X_{\texttt{trig}}$, which are listed in the Appendix. We run SAGGA until it discovers 1000 failing problems, then cluster these using k-means on SciBERT embeddings (Beltagy, Lo, and Cohan 2019) of each problem.

Table 4 shows three members from three discovered problem clusters, for the polynomial and trigonometric seeds. Intuitively, each cluster shows failures in a neighborhood around a prototypical problem – for instance, on $2x^{42} + \mathbf{k}$ the neural sequence integrator *correctly* integrates $2x^{42} + \mathbf{21}$, but not the problems in Cluster 2 (e.g. $2x^{42} + \mathbf{22}$). See Appendix Table 15 for more prototypes and the model's predictions.

Curiously, each problem in a neighborhood is governed by a common template – e.g. the problems $\{-104, -136, -33\}$ are governed by $\int k = kx + C$, yet the failures suggest that the neural sequence integrator has either not inferred the template, or does not apply it across the neighborhood. To investigate this phenomenon, we show the *raw* model prediction in Table 5, along with its simplified version and derivative. Compared to the underlying template $\int k = kx + C$ the model's raw output is long and complex. In contrast, the simplified version is short; we hypothesize this gap makes adhering to the template difficult.

Seed	Cluster 1	Cluster 2	Cluster 3
$X_{\texttt{poly}}$	-104	$2x^{42} + 22$	$-47 + 2/x - 2/x^{71}$
	-136	$2x^{42} + 28$	$-47 + 2/x - 31/x^{71}$
	-33	$2x^{42} + 68$	$-71 + 36/x + 2/x^{71}$
$X_{\texttt{trig}}$	$13\cos 19x$	$13\cos 83x - 59$	$10\sin 47x \cos 2x$
	$13\cos 83x$	$17\cos 37x - 49$	$10\sin 90x \cos 2x$
	$17\cos 47x$	$17\cos 41x - 45$	$19\sin 90x \cos 2x$

Table 4: Example robustness problems discovered by SAGGA which the neural sequence integrator fails to integrate. See Appendix Table 15 for model predictions.

x	Raw	Simplified	Deriv.
-104	$x^2 + 2x - (x + 25)^2$	$-48x - 625$	-48
-136	$x^2 - x(x + 130) + 2x$	$-128x$	-128
-33	$x^2 + x - (x + 16)^2$	$-31x - 256$	-31

Table 5: The raw model predictions for the problems $\mathbf{x}$ and their simplified forms. Each prediction is incorrect since its derivative is not equal to $\mathbf{x}$. The neural sequence integrator's raw output is long and varied compared to the underlying integration rule $\int k = kx + C$.

Discovering brittleness near validation problems. Finally, we use SAGGA to discover difficult problems that are close to a target set X – in our case validation problems – according to an explicit distance $d(\mathbf{x}, \tilde{\mathbf{x}})$. This allows for less hand-designing of the perturbations.

Specifically, we define a fitness which is high whenever a candidate is close to any problem in a target set X,

$$\text{fitness}(\tilde{\mathbf{x}}) = \left[\min_{\mathbf{x} \in X} d(\mathbf{x}, \tilde{\mathbf{x}})\right]^{-1} \cdot m(\tilde{\mathbf{x}}, f_\theta(\tilde{\mathbf{x}})). \quad (4)$$

We randomly sample 10 validation problems to form X, set SAGGA's initial seed to X, and use cosine similarity of SciBERT vectors to define the distance $d(\mathbf{x}, \tilde{\mathbf{x}})$. Since the *distance* now constrains the problems, we are free to use a wider set of mutations: changing a node's operation, adding an argument to a node, and replacing the node with a random constant, symbol, or simple operation.

Table 6 shows example problems that SAGGA discovers around the successful validation problems, exposing a wider class of robustness failures than our preceding experiments.

4 Integrating Parts But Not The Whole

While the preceding section identified weaknesses in robustness – for instance, integrating $26x^{42}$ but not $88x^{42}$ – a remaining question is whether successfully integrating a collection of primitives implies that a *composition* of those primitives will be successfully integrated.

Compositionality refers to forming compound equations from known primitives and operations. A compositional model should correctly integrate equations of the form,

$$f = f_1 \circ f_2 \circ \cdots \circ f_k, \quad (5)$$

Validation Problem	Nearby Failures
$-x^2 + x + \log(4)\tan(x)$	$-x^2 + x + \log(4)\tan(17x^2)$
	$-x^2 + x + \log(4)\tan(2x^2)$
	$-x^2 + x + \log(4)\tan(63/x^2)$
$\sqrt{3x+3}-2$	$\sqrt{-86^{x^2}+62/x}-40$
	$\sqrt{14+62/x}+4$
	$\sqrt{14+62/x}-2$
$\tan(\exp(2))/18x$	$\tan(\exp(2+71/x))/18x$
	$\tan(\exp(2-46^x))/18x$
	$\tan(\exp(37x))/18x$

Table 6: SAGGA discovers failures around successful validation problems, within a neighborhood defined by an explicit distance. Model predictions are in Appendix Table 15.

Input	Prediction	
$x^{1/3}$	$\frac{3}{4}x^{4/3}$	✓
$x^{1/606}$	$\frac{606}{607}x^{607/606}$	✓
$x^{1/3} + x^{1/606}$	$\frac{3}{5}x^{\frac{5}{3}} + \frac{6}{613}x^{\frac{613}{606}}$	✗
x^{209}	$\frac{1}{210}x^{210}$	✓
x^{764}	$\frac{1}{765}x^{765}$	✓
$x^{209} + x^{764}$	$\frac{1}{205}x^{205}$	✗
$14\cos(58x)$	$\frac{7}{29}\sin(58x)$	✓
$46\cos(84x)$	$\frac{23}{42}\sin(84x)$	✓
$14\cos(58x) + 46\cos(84x)$	$\sin(59x)\cos(x)$	✗

Table 7: *Compositionality* examples. We show the model's top prediction (beam search, width 10). The model successfully integrates the individual primitives, but not their sum.

where $f_1, \ldots, f_k$ are equations that the model successfully integrates, and $\circ$ is a binary operation (e.g. addition). For instance, a system that integrates x^2 and $\cos x$ and is capable of addition should successfully integrate $x^2 + \cos x$.

Formally, we say that a model is k-compositional with respect to equations X and operation $\circ$ when it successfully integrates any combination of k equations from X, $\sum_{\mathbf{x}\in\tilde{X}} m(\mathbf{x}, f_\theta(\mathbf{x})) = 0$, where $\tilde{X} = \{f_1\circ\cdots\circ f_k | f_i \in X\}$. We evaluate k-compositionality with respect to addition, using simple primitive functions and validation problems. As integration is linear, $\int(f+g) = \int f + \int g$, compositionality with respect to addition is a reasonable requirement.

Succeeding on simple primitives, failing on their sum. We collect simple primitives from the coefficient robustness experiments that the model successfully integrates (`coeff`), and successful exponents x^c or $x^{1/c}$, $c \in [0, 1000]$ (`exp`). We randomly sample 1000 compound equations $f_1 + \ldots + f_k$ for $k \in \{2, 3, 4\}$ and evaluate the failure rate. Table 8 shows the results. Adding two primitives gives failure rates of 29% and 85% for coefficient and exponent primitives, respectively, despite failing 0% of the time on the individual prim-

Type	Test	Fail@50	Fail@10	Fail@1
`exp(1)`	f_1	00.0	00.0	00.0
`exp(2)`	$f_1 + f_2$	70.8	72.4	84.9
`exp(3)`	$f_1 + f_2 + f_3$	91.3	97.5	99.5
`exp(4)`	$f_1 + \ldots + f_4$	86.2	97.4	99.8
`coeff(1)`	f_1	00.0	00.0	00.0
`coeff(2)`	$f_1 + f_2$	8.60	16.2	29.2
`coeff(3)`	$f_1 + f_2 + f_3$	23.8	37.5	61.0
`coeff(4)`	$f_1 + \ldots + f_4$	23.1	38.7	60.0
`valid(1)`	f_1	00.0	00.0	00.0
`valid(2)`	$f_1 + f_2$	6.80	14.5	15.0
`valid(3)`	$f_1 + f_2 + f_3$	21.5	36.5	43.6
`valid(4)`	$f_1 + \ldots + f_4$	52.5	69.0	79.2

Table 8: *Compositionality*. Top: **successful simple** primitives from the robustness experiments (Table 2). Bottom: **successful validation-set** primitives. Despite integrating each primitive, the model struggles to integrate their sums.

Nodes	Fail@10	Fail@1
1-15	0.4	1.6
20	1.9	10.7
25	7.2	17.2
30	24.4	37.1
35	49.0	59.2

Table 9: *Extrapolation* to more operator nodes under the training data generation process. Training used 1-15 nodes.

itives. As the number of compositions increases, the failure rate increases towards 100%. Table 7 shows examples.

Succeeding on test problems, failing on their sum. We perform a similar experiment using successful validation-set functions. We filter examples longer than 20 tokens so that composed equations are within the training domain in terms of length, and sample 1000 compound equations $f_1 + \ldots + f_k$ for $k \in \{2, 3, 4\}$. As seen in Table 8, the failure rate grows as the number of compositions increases, similar to the simple primitives case. Maximizing the likelihood of a large training set did not yield a compositional model.

5 Departing Further From Training

The preceding experiments found problems that were nearby to, or composed directly from, in-distribution examples. In this section, we deliberately move from the model's training distribution to evaluate its *out-of-distribution* generalization. First, we study *extrapolation* to longer equation sizes than those in its training distribution, and to integer ranges that are only sparsely covered in the training set. Then we use SAGGA to expose exotic failures and reveal problem classes that were not covered during training.

Longer problems are more difficult. First, we use the same data-generating process as for training, but *vary its parameters* to depart from the training distribution. Specifically, we test extrapolation on number of operator nodes in
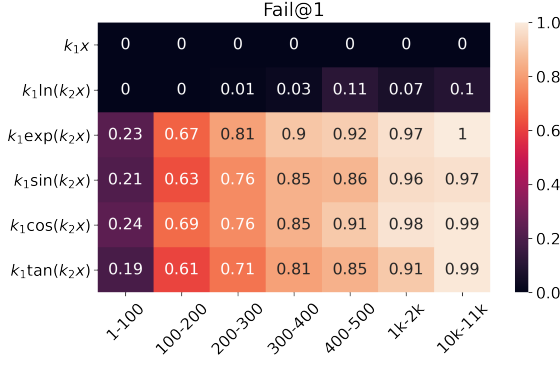
Figure 2: *Integer extrapolation.* Failure rates for integrating simple primitives with coefficients from the specified range.

Problem	Exploit
$169\sin(4x)/x$	Uses $\mathrm{Si}(\cdot)$.
$-2\sin(42/x)$	Uses $\mathrm{Ci}(\cdot)$.
$-2\sin(185x^2)\cos(2x)$	Uses Fresnel S, C integrals.
$357^{x^{2^x}}+2\sin(2x)$	Uses incomplete gamma $\Gamma(a,x)$.
$1/(x^{48}(3x+2)^{49})$	Decoding does not terminate.

Table 10: *Exploits* discovered by SAGGA whose integrals use out-of-domain functions. See Appendix Table 17 for model predictions.

each equation tree, using Lample and Charton (2019)'s data generation process and varying the `max_ops` parameter. Table 9 shows performance when `max_ops` is increased past the model's training domain (1-15). The neural sequence integrator does show some extrapolation to equation trees with more operator nodes than it was trained on, but its failure rate increases substantially as the number of nodes increases.

Larger failures on larger digits. Next, we study performance as integer values increase, quickly going out-of-domain. Considering a sample of 200,000 sequences from the training distribution, 99.4% of the positive integers were between 1 and 100. Other ranges were non-empty but sparsely represented; for instance, 0.2% of the integers were between 100 and 200, and 0.096% between 1,000 and 2,000. Figure 2 shows performance on primitive functions with coefficients from the specified range. As in the robustness experiments, the x and $\ln$ primitives perform well, showing that there is *some* ability to use large numbers. However, performance severely degrades for the `exp`, `sin`, `cos`, `tan` primitives as the coefficient magnitudes increase, reaching near 100% failure rates on large coefficients.

Discovering unsupported functionality. Next, we run SAGGA in an unconstrained setting with all mutation types, favoring short problems using the fitness, $F(f_\theta, \mathbf{x}) = m(\mathbf{x}, f_\theta(\mathbf{x})) \cdot \frac{1}{|\mathbf{x}|}$, which is positive when the model returns an incorrect integral for $\mathbf{x}$, and higher for shorter problems.

SAGGA discovers *exploits* based on the neural sequence integrator's limited training distribution, such as problems whose integral is expressed using the Gamma function $\Gamma(\cdot)$,

Cluster 1	Cluster 2	Cluster 3	Cluster 4
119^x	$-240^x+2\cos 2x$	-100^{x^x}	$158x^{x^2}+611$
132^x	$-398^x+2\cos 2x$	-149^{x^x}	$256x^{x^2}+191$
136^x	$-692^x+2\sin 2x$	-151^{x^x}	$332x^{x^2}+559$

Table 11: SAGGA discovers many failures that involve x in an exponent. See Appendix Table 16 for model predictions.
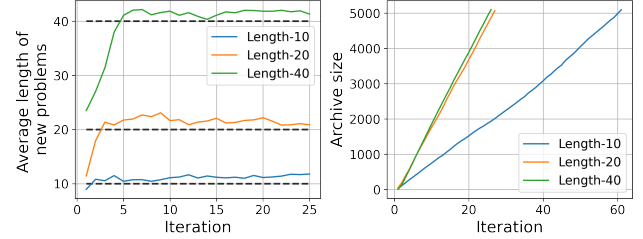

Figure 3: Left: Average length of discovered problems in each iteration, using fitness functions that promote the given length. SAGGA discovers problems of each target length. Right: SAGGA discovers longer problems at a higher rate.

or the cosine integral Ci, which are not included in its training data (Table 10).[3] These examples are a reminder that the sequence-to-sequence paradigm determines which functions are 'built in' by inclusion in training data; omitted behavior is left unspecified, leaving it susceptible to exploits.

Finally, the last problem in Table 10 caused the neural sequence integrator to enter a non-terminating loop during decoding (Appendix Table 17), a known idiosyncrasy of autoregressive models with beam search (Welleck et al. 2020).

SAGGA also finds many clusters that indicate the neural sequence integrator struggles when x appears in an exponent. The discovered problems in Table 11 are a microcosm of our previous findings: For the first cluster, we manually found a nearby problem, 30^x, that the model gets correct; this cluster is a *robustness* failure. The second cluster shows how such failures cascade further as the function is *composed*. The final two clusters involve x^x or x^{x^2}, which do not have analytical integrals;[4] these clusters are *exploits*.

Finding problems with target properties. Finally, we generate failures of a target length, by running SAGGA to target length 10, 20, and 40 problems. As seen in Figure 3, SAGGA converges to find problems of the target length. Based on our extrapolation experiments, we expect SAGGA to fail more often on longer equations. The right-hand plot in Figure 3 shows that it is also easier to *find* failures for longer equations, in that the archive grows more quickly for longer target lengths. While we visually inspect short equations for interpretability, the growth rate is a reminder that the space of failures is vast for longer equations.

---

[3] https://en.wikipedia.org/wiki/Trigonometric_integral
[4] https://www.wolframalpha.com/input/?i=integral+of+x**x

Beam size	Fail@Beam	gt>max(beam)	gt>min(beam)
1	7.2	0.6	0.59
5	5.0	0.1	1.1
10	4.6	0.1	1.6

Table 12: Percentage of cases on the `FWD` validation set for which none of the beam candidates are correct and the ground truth is scored higher than the entire beam (gt>max(beam)), or any beam candidate (gt>min(beam)). With perfect search, at least $\frac{3}{5}$ of the failures would remain.

6 Is it a search problem?

Both the experiments of Lample and Charton (2019) and our own generate candidate solutions from a sequence-to-sequence model using beam search. The model is trained to assign high scores to correct solutions, yet beam search is an approximate method, which may not find the highest-scoring sequence. This raises the possibility that failures are due to search rather than the model: what if the highest scoring sequences are correct, but not found by beam search?

To investigate this, we experiment on problems with a well-formatted ground-truth integral, using problems from the `FWD` validation set from Lample and Charton (2019). On failure cases, we first test *whether the ground-truth would be ranked at the top of the beam if beam search had discovered it*, checking whether the ground-truth's score is higher than those of all beam candidates (**gt > max(beam)**). Since the search-and-verify setting only requires the correct integral to be *in* the beam rather than at the top, we also check whether the ground-truth would be in the existing beam if found (**gt > min(beam)**). The results are in Table 12. In at least $\frac{3}{5}$ of errors, the ground truth is scored below all beam hypotheses, meaning that these failures would remain with perfect search; search is not the only problem on these examples.

A related question for the search-and-verify setting is whether increasing the search budget – the number of candidates that are returned and verified – eliminates failures. While verifying more candidates does reduce failure rate, i.e. Fail@k decreases as k increases, our experiments do not indicate failures approach zero for practical search budgets. For instance, in our compositionality `exp` experiments (Table 8), verifying 50 instead of 1 candidate reduces failures, but not below 90%. Larger search budgets quickly become impractical to verify: for compositionality `exp`, the verification time increases substantially, from around 5 minutes for 1 candidate to over 2 hours for 50, with worst-case verification time of 41.6 hours on these simple problems (Table 13). Improving the underlying model remains a main issue.

7 Related Work

In this work, we study systematic generalization in sequence models applied to symbolic integration, in terms of robustness, compositionality, and extrapolation, and develop a genetic algorithm for building adversarial problem sets.

Symbolic mathematics and sequence models.   Several works study extrapolation to longer sequences and larger digits in synthetic arithmetic and basic mathematics tasks (Zaremba and Sutskever 2014; Trask et al. 2018; Saxton et al. 2019; Nogueira, Jiang, and Li 2021). Sequence models have also been applied to polynomial rewriting (Piotrowski et al. 2019; Agarwal, Aditya, and Goyal 2021), and differential system stability (Charton, Hayat, and Lample 2021). For symbolic integration, Davis (2019) argue that the neural sequence integrator's test performance should be qualified, though without an empirical demonstration. These critiques motivate our focus on the neural sequence integrator (Lample and Charton 2019), whose performance we characterize and empirically study in terms of systematic generalization.

Systematic generalization.   Several works identify difficulties with modern methods on synthetic tasks (e.g. Lake and Baroni (2018); Bahdanau et al. (2019); Hupkes et al. (2020); Kim and Linzen (2020)) and machine translation (Raunak et al. 2019), with a focus on compositionality and extrapolation. Some methods address systematicity with inductive biases in model structure (Andreas et al. 2016; Bahdanau et al. 2019), and others through the data (Hill et al. 2020; Andreas 2020) or learning procedure (Lake 2019; Vani et al. 2021). We focus on systematic generalization deficiencies in a state-of-the-art model in a new setting – symbolic integration – with additional aspects of generalization.

Robustness and adversaries in sequence models.   Several works study robustness in NLP, including classification (Tu et al. 2020), word substitutions (Jia et al. 2019), domain shift in QA (Kamath, Jia, and Liang 2020) or topic distributions (Oren et al. 2019). Several methods find adversarial examples in NLP (Morris et al. 2020). Alzantot et al. (2018) use genetic algorithms in a classification setting, while we consider generation. Michel et al. (2019) constrain input sequences to be similar and use a gradient-based attack to swap tokens. We face a non-differentiable cost and generate large collections of failures with a wide class of mutations.

8 Conclusion

We study generalization in symbolic mathematics using the predominant modeling paradigm: a large-scale transformer trained with maximum likelihood. We find deficiencies that are not captured by test accuracy, including brittleness to small perturbations, difficulty composing known solutions, and gaps in the training distribution. We offer speculations based on our results. Due to the large space of equations, practical empirical distributions do not provide a dense sampling of individual problem types (e.g. $k_1 \cos(k_2 x)$), and each empirical sample contains shared biases from the underlying data generator (e.g. integer values, lengths). Thus, sparse test sets do not adequately measure systematic generalization. From a learning perspective, generic networks trained with SGD do not necessarily favor the simplest hypothesis to explain the data; thus a sparse training set yields an underconstrained hypothesis space, with hypotheses that may not strongly generalize (e.g. Table 5), causing behavior that breaks simple rules (e.g. adhering to a template or following the sum rule). We suspect that inductive biases– e.g. encoded through the training distribution, architectural components, or learning algorithm– are needed to narrow hypotheses to those that strongly generalize.

References

Agarwal, V.; Aditya, S.; and Goyal, N. 2021. Analyzing the Nuances of Transformers' Polynomial Simplification Abilities. arXiv:2104.14095.

Agrawal, A.; Batra, D.; and Parikh, D. 2016. Analyzing the Behavior of Visual Question Answering Models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 1955–1960. Austin, Texas: Association for Computational Linguistics.

Alzantot, M.; Sharma, Y.; Elgohary, A.; Ho, B.-J.; Srivastava, M.; and Chang, K.-W. 2018. Generating Natural Language Adversarial Examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2890–2896. Brussels, Belgium: Association for Computational Linguistics.

Andreas, J. 2020. Good-Enough Compositional Data Augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7556–7566. Online: Association for Computational Linguistics.

Andreas, J.; Rohrbach, M.; Darrell, T.; and Klein, D. 2016. Neural module networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN 9781467388504.

Bahdanau, D.; Murty, S.; Noukhovitch, M.; Nguyen, T. H.; De Vries, H.; and Courville, A. 2019. Systematic generalization: What is required and can it be learned? In *7th International Conference on Learning Representations, ICLR 2019*.

Beltagy, I.; Lo, K.; and Cohan, A. 2019. SciBERT: A Pretrained Language Model for Scientific Text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3615–3620. Hong Kong, China: Association for Computational Linguistics.

Charton, F.; Hayat, A.; and Lample, G. 2021. Learning advanced mathematical computations from examples. arXiv:2006.06462.

Davis, E. 2019. The Use of Deep Learning for Symbolic Integration: A Review of (Lample and Charton, 2019). arXiv:1912.05752.

Gardner, M.; Artzi, Y.; Basmov, V.; Berant, J.; Bogin, B.; Chen, S.; Dasigi, P.; Dua, D.; Elazar, Y.; Gottumukkala, A.; Gupta, N.; Hajishirzi, H.; Ilharco, G.; Khashabi, D.; Lin, K.; Liu, J.; Liu, N. F.; Mulcaire, P.; Ning, Q.; Singh, S.; Smith, N. A.; Subramanian, S.; Tsarfaty, R.; Wallace, E.; Zhang, A.; and Zhou, B. 2020. Evaluating Models' Local Decision Boundaries via Contrast Sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1307–1323. Online: Association for Computational Linguistics.

Henighan, T.; Kaplan, J.; Katz, M.; Chen, M.; Hesse, C.; Jackson, J.; Jun, H.; Brown, T. B.; Dhariwal, P.; Gray, S.; Hallacy, C.; Mann, B.; Radford, A.; Ramesh, A.; Ryder, N.; Ziegler, D. M.; Schulman, J.; Amodei, D.; and McCandlish, S. 2020. Scaling Laws for Autoregressive Generative Modeling. *ArXiv*, abs/2010.14701.

Hill, F.; Lampinen, A.; Schneider, R.; Clark, S.; Botvinick, M.; McClelland, J. L.; and Santoro, A. 2020. Environmental drivers of systematicity and generalization in a situated agent. In *International Conference on Learning Representations*.

Holtzman, A.; Buys, J.; Du, L.; Forbes, M.; and Choi, Y. 2020. The Curious Case of Neural Text Degeneration. In *International Conference on Learning Representations*.

Hupkes, D.; Dankers, V.; Mul, M.; and Bruni, E. 2020. Compositionality Decomposed: How do Neural Networks Generalise? *Journal of Artificial Intelligence Research*.

Jia, R.; Raghunathan, A.; Göksel, K.; and Liang, P. 2019. Certified Robustness to Adversarial Word Substitutions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 4129–4142. Hong Kong, China: Association for Computational Linguistics.

Kamath, A.; Jia, R.; and Liang, P. 2020. Selective Question Answering under Domain Shift. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 5684–5696. Online: Association for Computational Linguistics.

Kim, N.; and Linzen, T. 2020. COGS: A Compositional Generalization Challenge Based on Semantic Interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 9087–9105. Online: Association for Computational Linguistics.

Lake, B.; and Baroni, M. 2018. Still Not Systematic After All These Years: On the Compositional Skills of Sequence-To-Sequence Recurrent Networks. *Iclr2018*.

Lake, B. M. 2019. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems*.

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

Michel, P.; Li, X.; Neubig, G.; and Pino, J. 2019. On Evaluation of Adversarial Perturbations for Sequence-to-Sequence Models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 3103–3114. Minneapolis, Minnesota: Association for Computational Linguistics.

Morris, J.; Lifland, E.; Yoo, J. Y.; Grigsby, J.; Jin, D.; and Qi, Y. 2020. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 119–126. Online: Association for Computational Linguistics.

Nogueira, R.; Jiang, Z.; and Li, J. J. 2021. Investigating the Limitations of the Transformers with Simple Arithmetic Tasks. *ArXiv*, abs/2102.13019.

Oren, Y.; Sagawa, S.; Hashimoto, T. B.; and Liang, P. 2019. Distributionally Robust Language Modeling. In

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 4227–4237. Hong Kong, China: Association for Computational Linguistics.

Piotrowski, B.; Urban, J.; Brown, C. E.; and Kaliszyk, C. 2019. Can Neural Networks Learn Symbolic Rewriting? *CoRR*, abs/1911.04873.

Raunak, V.; Kumar, V.; Metze, F.; and Callan, J. 2019. On Compositionality in Neural Machine Translation. *ArXiv*, abs/1911.01497.

Saxton, D.; Grefenstette, E.; Hill, F.; and Kohli, P. 2019. Analysing Mathematical Reasoning Abilities of Neural Models. In *International Conference on Learning Representations*.

Trask, A.; Hill, F.; Reed, S.; Rae, J.; Dyer, C.; and Blunsom, P. 2018. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*.

Tu, L.; Lalwani, G.; Gella, S.; and He, H. 2020. An Empirical Study on Robustness to Spurious Correlations using Pre-trained Language Models. *Transactions of the Association for Computational Linguistics*, 8: 621–633.

Vani, A.; Schwarzer, M.; Lu, Y.; Dhekane, E.; and Courville, A. 2021. Iterated learning for emergent systematicity in {VQA}. In *International Conference on Learning Representations*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*.

Wallace, E.; Feng, S.; Kandpal, N.; Gardner, M.; and Singh, S. 2019. Universal Adversarial Triggers for Attacking and Analyzing NLP. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2153–2162. Hong Kong, China: Association for Computational Linguistics.

Welleck, S.; Kulikov, I.; Kim, J.; Pang, R. Y.; and Cho, K. 2020. Consistency of a Recurrent Language Model With Respect to Incomplete Decoding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 5553–5568. Online: Association for Computational Linguistics.

Zaremba, W.; and Sutskever, I. 2014. Learning to Execute. *CoRR*, abs/1410.4615.