

Enhancing Sequence Model with Mathematical Reasoning in Symbolic Integration

Xingqi Lin¹, Liangyu Chen¹(✉), Zhengfeng Yang¹ and Zhenbing Zeng²

¹ Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China

² Department of Mathematics, Shanghai University, Shanghai 200444, China
lychen@sei.ecnu.edu.cn

Abstract. Sequence model has shown its efficiency in tackling integration problems, outperforming traditional mathematical software on specific datasets. However, it also encounters some challenges: robustness against minor perturbations, compositionality for decomposable operations, and out-of-distribution generalization when dealing with larger values, longer problems, and functions not covered in the training set. These issues arise from the fact that integration problems can only be partially regarded as a language translation task because integration follows its own mathematical rules. To address the above issues, this paper proposes a novel approach that enhances sequence model with mathematical reasoning. We introduce the abstraction of coefficients, perform expression decomposition, and substitute known functions for unknown counterparts. Our model achieves 83.6% accuracy in integration testing, 100% accuracy in robustness testing and 100% accuracy in additive composite expressions. By the mathematical rewriting, it also exhibits notable performance in extrapolation beyond the distribution. Moreover, our model passes the SAGGA test. In general, we obtain a robust symbolic integrator.

Keywords: Sequence Model, Deep Learning, AI Mathematics, Indefinite Integral.

1 Introduction

With the rapid development of artificial intelligence, deep learning has recently achieved significant breakthroughs in various fields, such as computer vision, natural language processing, and pattern recognition. These remarkable achievements have led researchers to focus on other fields, such as mathematics, which was previously more reliant on human intelligence. LC model is a pioneering work, which uses neural networks to solve mathematical problems of symbol integration[11]. LC model converts mathematical expressions into sequences and considers the integration problem as a sequence-to-sequence translation task, where the goal is to translate one sequence (the original expression) into another sequence (the indefinite integral expression). Although LC model has no mathematical knowledge, it achieves near-perfect test accuracy. More specifically, LC model outperforms several commercial symbolic

computation software systems (such as Maple, Matlab, and Mathematica) and can even solve indefinite integration problems that commercial software systems cannot handle.

LC model has sparked heated discussions and debates. Davis [4] questioned the high accuracy of the model and doubted that it might benefit from bias in the test set and the lack of special functions. Noorbakhsh et al. [15] used pre-trained large language models with fine-tuning strategies to effectively reduce training costs. Welleck et al. [22] evaluated the generalization of LC model and found that it performs poorly in terms of robustness, compositionality, and out-of-distribution generalization. Specifically, LC model (1) is fragile to small perturbations (for example, it can correctly solve $\int 26x^{42}dx$ but fails to integrate $\int 53x^{24}dx$, which should follow the same rule $\int k_1 x^{k_2} dx = \frac{k_1}{k_2+1} x^{k_2+1} + C$ for all constants k_1, k_2); (2) struggles to combine known solutions, not conforming to the rule $\int f_1 dx + \int f_2 dx = \int (f_1 + f_2) dx$; (3) has difficulty in generalizing to longer expressions, larger numbers, and special functions.

From the above issues, it is evident that integration problems can only be partially regarded as a language translation task because integration follows its own mathematical rules. Therefore, this paper extends LC model and enhances sequence model with mathematical reasoning in symbolic integration. In summary, our main contributions are listed as follows:

- Numbers in mathematical expressions are abstracted with coefficients. This approach resolves the model's brittleness issue in simple primitive functions.
- To achieve compositionality, expressions are decomposed under mathematical rules and then processed separately by neural networks. All mathematical operations that can be decomposed are suitable for this method, and we take addition as an example.
- Out-of-distribution generalization is enhanced by employing mathematical transformations to handle data beyond the training set, which means that our model can solve the integral problem with new mathematical functions.

The rest of the paper is organized as follows. We discuss the related work in Section 2 and introduce the methodology in Section 3. Experimental results are provided in Section 4. Finally, the paper is concluded in Section 5.

2 Related Work

As early as the 1980s, the application of artificial neural networks (ANN) to matrix inverse and Fourier transform was first proposed [13]. Subsequently, neural networks were also applied to solve differential equations [12,3,9]. However, these researches were not put into practical use due to the limited computing power and the complexity of neural networks.

With the development of computer technology, neural networks have been quickly developed and have been increasingly applied in mathematical domains, such as mixed boolean-arithmetic expressions [6], logical reasoning [5], boolean satisfiability problem (SAT) [18], and so on [20,14,16]. In 2020, Lample and Charton proposed the application of the *transformer* in two problems of symbolic calculation: function

integration and ordinary differential equations [11]. Their work is the beginning of deep learning for symbolic integration problems.

Davis [4] outlined five issues about LC model. (1) In most cases, the integral of a nontrivial elementary function is not an elementary function itself. What's worse, the question of whether the integral of an elementary function is also an elementary function is, in principle, undecidable [17]. The integration from elementary to non-elementary functions is impossible with LC model but can be achieved with Mathematica. (2) The training and test set of LC model are self-generated and do not include non-simplified expressions. (3) The vocabulary of LC model lacks special functions common in differential problems. So, LC model is limited to a much smaller space of the elementary functions than Mathematica when searching for solutions. (4) For problems that cannot be solved, LC model will provide an incorrect answer while Mathematica will either stop the calculation or show time out. Of course, the preference between the wrong and the non-response is subjective. (5) The test set of LC model has significant built-in biases, which is unfair to Mathematica.

In addition, it is noteworthy that LC model requires a large amount of training data to achieve high accuracy since it is based on the transformer architecture [21]. To solve this issue, there is a two-step approach: first training a transformer-based model for language translation tasks and then fine-tuning the pretrained model to solve the downstream task of symbolic mathematics [15]. Noorbakhsh et al. used the mBART (and Marian-MT) transformer and explored the impact of different multilingual tasks on integration accuracy, such as English to Romanian, English to Arabic, and French to English. The enhanced model achieves comparable accuracy on the integration task, using around 1.5 orders of magnitude fewer training samples than LC model.

The generalization of LC model beyond the test set needs evaluation. Unfortunately, LC model has challenges in achieving robustness, compositionality, and out-of-distribution generalization [22]. That is to say, despite the perfect accuracy of LC model, it has some flaws that include vulnerability to minor perturbations, difficulties in combining known solutions, inability to handle longer problems, larger values, and functions not covered in the training set. Consequently, further research is required to enhance the generalization of LC model.

3 Methodology

3.1 Problem Formulation

The problem of solving indefinite integrals can be stated as follows: given a mathematical expression f , find its indefinite integral F with respect to the variable x , denoted as $F = \int f dx$. For example, $\int 2x dx = x^2 + C$ where C is a constant number. To be more concise, we will omit C in the following discussion.

Although there are some symbolic computation software systems like Maple, Matlab, and Mathematica now, improvement is still possible, such as faster execution or effective solutions for complex problems. The neural sequence integrator, LC model [11], achieves high accuracy in its own generated test set (even surpassing the commercial computation software mentioned above). Nevertheless, test accuracy alone does

not guarantee powerful generalization. According to [22], the following content lists three vulnerability aspects of LC model.

Robustness is the ability of a system or algorithm to maintain stable performance and functionality in the presence of uncertainties, variations, or unexpected conditions. For symbolic integrator, robustness refers to whether integrating an in-distribution problem means success on nearby problems being governed by the same underlying rule. Even when ignoring more complex situations, the robustness of LC model on simple primitive functions is relatively poor, leading to some ridiculous errors. For instance, when solving $\int k_1 \sin k_2 x dx$, the correctness of the answer varies as k_1 and k_2 change, which is unreasonable.

Compositionality refers to the property of a system or model where complex expressions or structures can be formed by combining simpler, known components or elements through specified operations. A compositional integral model should correctly integrate

$$f = f_1 \circ f_2 \circ \dots \circ f_k, \quad (1)$$

where f_1, f_2, \dots, f_k are expressions the model can correctly handle, and \circ is a binary operation (e.g., addition). Theoretically, if the model succeeds in a collection of problems, it should be able to solve the composition of those problems. For example, a system that successfully integrates x and $\sin x$, should correctly integrate $x + \sin x$. Furthermore, for indefinite integration, the rule $\int f_1 + \int f_2 = \int (f_1 + f_2)$ exists. Thus, it is reasonable to expect LC model to possess additive compositionality. However, the reality is quite different. As the number of compositions increases, the accuracy of LC model decreases sharply.

Out-of-distribution generalization refers to the ability of the model to handle data that are not in the training set [1,10,2,7]. Generalizing to out-of-distribution data is crucial for integral models since real-world mathematical expressions are diverse and impossible to be completely included in the training data. However, the performance of LC model is not satisfactory in this case. The range of numbers in the training set is limited to $[-10,10]$, and as the integers in the problem increase, the accuracy of LC model decreases. Additionally, LC model struggles to solve problems with more than 30 operators. Moreover, since LC model only considers elementary functions, it becomes powerless when encountering special functions not present in its vocabulary.

Therefore, the problem this paper aims to address is: how to enhance the model's generalization capabilities in robustness, compositionality, and out-of-distribution data without compromising accuracy and then establish a more reasonable neural sequence integrator.

3.2 Framework

In response to the above problem, we propose an approach to enhance the sequence model with mathematical reasoning. LC model is a general text sequence model that does not understand the components in mathematical expressions such as coefficients, variables, integers, and operators. Consequently, we extract individual parts of

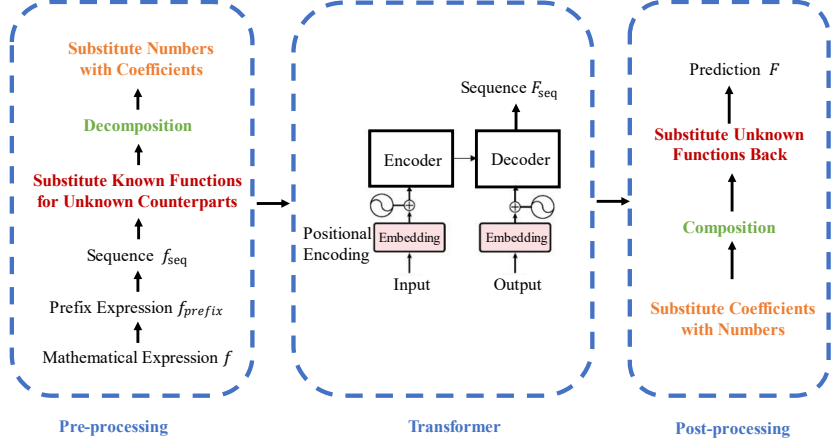


Fig. 1. The architecture of our model Rosymtor. The model consists of three parts: pre-processing, transformer, and post-processing. The abstraction of coefficients is highlighted in orange, decomposition and composition in green, and mathematical substitution in red.

expressions for special processing. First, we substitute unknown special functions with known primitive functions according to mathematical rules, avoiding unknown words into the sequence model. Moreover, we decompose expressions based on decomposable operators which can divide them into multiple components equally in mathematics, employing the divide-and-conquer strategy for longer expressions to enhance the efficiency and accuracy of integration problems. Lastly, while coefficients subtly impact solving mathematical problems, they significantly influence the sequence model. Thus, we use parameters to replace coefficients in the expression to improve generalization.

In summary, we present an improved model called Rosymtor (**R**obust **S**ymbolic **I**ntegrator), whose framework is depicted in Figure 1, consisting of the following three parts:

- **Pre-processing of mathematical expression.** To start with, we transform an original mathematical expression into a formal sequence. Then, the abstraction of coefficients is introduced for robustness and additive decomposition for compositionality. Out-of-distribution generalization is taken into consideration too. This part combines machine translation and symbolic computation techniques, laying the foundation for follow-up steps.
- **Transformer neural network.** The input of the transformer part is a sequence derived from a mathematical expression. Then the transformer utilizes an encoder-decoder architecture to translate this sequence. By minimizing the loss computed with predicted output and actual results, the model learns the integration rules and techniques for solving indefinite integrals.
- **Post-processing of prediction.** After the transformer generates a sequence represented by indices, we map them to corresponding words and obtain a prefix

expression. The coefficients in the prefix expression are replaced with their matching numbers. Besides, addition combinations and mathematical transformations are performed when necessary.

In the part of pre-processing of mathematical expression, these expressions can be represented as trees, with operators as internal nodes and numbers or variables as leaves. A pre-order traversal of a tree generates a computer-friendly prefix expression, shown in Table 1. During parsing, the words “INT+” and “INT-” are introduced as prefixes for numbers. For example, “-5” is parsed as [INT-, 5], while “108” is parsed as [INT+, 1, 0, 8]. For more details such as tokenizer, refer to [11].

Table 1. Infix and prefix notation of mathematical expression.

Infix	Prefix
$a_0 \times (x + \ln(x))$	mul a_0 add x ln x
e^{a_0-x}	exp add a_0 mul INT- 1 x
$\ln(x) + \tan(x)$	add ln x tan x
$a_0 \times x^2$	mul a_0 pow x INT+ 2

This paper concentrates on the processing of mathematical expressions, with the abstraction of coefficients elaborated in Section 3.3, decomposition and composition in Section 3.4, and the out-of-dataset situation in Section 3.5.

3.3 Abstraction of Coefficients

To improve robustness, this paper introduces the abstraction of coefficients, which involves two steps highlighted in orange in Figure 1: substitute numbers with coefficients and substitute coefficients with numbers. The process is illustrated in Figure 2.

Substitute numbers with coefficients. Based on the number parsing, we can implement coefficient substitution. Specifically, the infix expression is first converted into a prefix expression, and then the prefix expression is traversed from left to right. When encountering “INT+” or “INT-”, the following numbers are read, and the coefficients a_0, a_1, a_2, \dots are sequentially used to replace the numbers while simultaneously recording the mapping between coefficients and numbers.

To adapt to the expressions with abstracted coefficients, we redesign the model’s training dataset. Many expressions with coefficients are generated as training data, which differ from LC model, whose expressions only contain numbers. It is worth noting that although the generation probabilities for variables (i.e., x) and coefficients (i.e., a_0, a_1, a_2, \dots) are both 50%, and the probability for numbers is set to 0, numbers inevitably appear during expression simplification. Therefore, the integrator also needs to handle numbers.

Substitute coefficients with numbers. After substituting coefficients for numbers, expressions are input into the trained transformer for solving. Results are then replaced with the corresponding numbers instead of coefficients. Finally, expressions are

simplified and converted back to infix form. At this point, we obtain the answer for the integral.

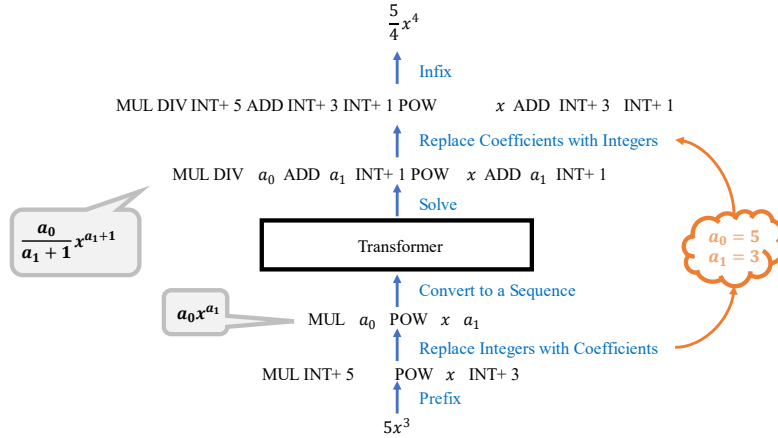


Fig. 2. Abstraction of coefficients in our model.

3.4 Decomposition and Composition

According to the compositionality of integration, we adopt the divide-and-conquer strategy for long mathematical expressions, which involved two steps in Figure 1: Decomposition and Composition.

Decomposition. We decompose longer expressions into smaller components to achieve compositionality before they enter the neural network. This process doesn't impact the result in mathematics but reduces the computational complexity. Taking the addition decomposition in integral as an example. The original integrand is decomposed into several parts according to the addition rule.

Composition. Every decomposed part is solved separately, and the results are collected and combined. The concatenated result is then simplified to obtain the final answer. The process of decomposing expressions by addition splits complex expressions into several relatively simpler parts, further simplifying the structure of expressions.

3.5 Out-of-distribution Generation

For sequence models, unknown words that are out of the training vocabulary will drastically decrease the translation performance. As to the integral task, if the input sequence contains unknown functions, the model is hard to handle them correctly. Inspired by the word synonym, we design the rewriting module to substitute the unknown function with the composition of other known functions, namely, two steps highlighted in red in Figure 1.

Substitute known functions for unknown counterparts. For instance, focusing on trigonometric functions, there exists the formula $\tan(x) = \sin(x) / \cos(x)$. If the transformer model does not know the function *tan* but knows *sin* and *cos*, it can learn the transformation principle of *tan* during training and then handle integrals involving *tan*. Therefore, as long as we provide the transformer with mathematical rewriting rules, the model can handle unknown functions.

Substitute unknown functions back. After generating predictions by the transformer model, which may include unknown words, it is necessary for post-processing. For example, upon identifying the structural pattern *sin/cos*, we should substitute it with *tan*. As a result, the final result may include unknown words. This approach enables out-of-distribution generalization without expanding the predefined vocabulary.

4 Experiments

In this section, we conduct experiments to compare the performance of Rosymtor and LC model. Specifically, we answer the following four research questions.

- RQ1: How do Rosymtor and LC model perform on a generic test set?
- RQ2: How is the robustness of Rosymtor and LC model?
- RQ3: How is the compositionality of Rosymtor and LC model?
- RQ4: How is the out-of-distribution generalization of Rosymtor and LC model?

We describe the datasets in Section 4.1 and introduce some implementation details of the experiments in Section 4.2. Afterward, we present the explanations to answer the above research questions in Section 4.3 ~ 4.6 respectively. Code is available at <https://anonymous.4open.science/r/Rosymtor-8EAB/>.

4.1 Datasets

In our experiments, the training, validation, and test sets are all generated by the method of backward generation provided by [11]. As is well known, integrating long expressions is time expensive, while differentiation is relatively simpler. Therefore, backward generation is to produce a mathematical expression F and then compute its derivative f . Afterward, the pair (f, F) is included in the dataset.

The validity of mathematical expressions is not considered when generation, resulting in a few invalid expressions, which we should filter in advance. Excessive nesting is so challenging that expressions with a maximum nesting level greater than 4 are regarded as invalid. Besides, components like $\sqrt{-3}$ and $\ln(-5)$ are meaningless in elementary mathematics which are also excluded from the dataset. After data cleaning, the dataset is divided into three parts, namely, 10,000 for validation, 10,000 for testing, and 10,000,000 for training.

The datasets in this work are different from those in LC model, shown in Table 2. The probability of leaf nodes being coefficients in the LC model is 0, whereas 50% in Rosymtor. The size of our training set is smaller than that of LC model because the abstraction of coefficients reduces expression redundancy. Our training set includes all the numbers, operators, and functions in LC model.

Table 2. The comparison between the training sets of LC model and Rosymtor.

	LC model	Rosymtor
Size of dataset	40 million	10 million
Storage Size	14.4GB	2.6GB
Max length of equations	512	512
Probability of coefficients	0	0.5
Probability of numbers	0.25	0
Probability of variables	0.75	0.5

4.2 Implementation Details

Sympy is a Python library for symbolic mathematics. Infix expressions shown in Table 1 are essentially strings without mathematical meaning. We convert these infix expressions into Sympy expressions to perform simplification, expansion, integration and differentiation. By the way, infix expression, prefix expression, and Sympy expression can be mutually converted in pairs.

As to the model implementation, we use Adam [8] to optimize our model. Similar to LC model, Rosymtor has 8 attention heads, 6 layers, and a dimensionality of 512. We set the learning rate to 0.0001 and the batch size to 32.

All models are implemented by PyTorch 1.12 using Python 3.7, and all experiments are executed on a CentOS Linux server with the main configuration of GPU RTX 2080Ti, CPU@3.60GHz, 8GB RAM, and 1TB SSD Disk.

4.3 Evaluation of Accuracy

In this experiment, four different models are tested and compared. They are: (1) the LC model; (2) LCAC, the LC model plus the abstraction of coefficients; (3) LCDC, the LC model plus the decomposition and combination; and (4) our model Rosymtor, the LC model plus the abstraction of coefficients, the decomposition and composition, and out-of-distribution generation.

The results are shown in Table 3. The LCAC and LC models have approximately equal accuracy, indicating that coefficient substitution has little impact on accuracy. The LCDC model has minor improvement in accuracy because decomposing complex expressions into several simpler ones makes it easier for the integrator to work. The Rosymtor model further improves accuracy, which is reasonable: coefficient substitution alone may lead to too many coefficients, and decomposition in advance can partially address this issue. Overall, coefficient substitution and additive decomposition have a limited impact on accuracy; they are designed to enhance generalization.

Table 3. Accuracy comparison among four models.

	LC	LCAC	LCDC	Rosymtor
Accuracy	80.4%	79.0%	82.3%	83.6%

4.4 Evaluation of Robustness

Next, we test robustness on simple primitive functions such as $k_1 \ln(k_2 x)$, $k_1 x$, $k_1 \cos(k_2 x)$, etc., where k_1 and k_2 are random integers ranging from 1 to 100. Table 4 compares the accuracy of the models before and after coefficient substitution. Higher accuracy indicates better robustness.

From a positive perspective, LC model demonstrates robustness on $k_1 \ln(k_2 x)$, as it learns the rule $\int k_1 \ln(k_2 x) dx = k_1 x (\ln(k_2 x) - 1)$. However, on other primitive functions, especially trigonometric functions, the performance of LC model is poor. In comparison, Rosymtor perfectly solves integrals for all primitive functions. Note that 100% accuracy is mathematically impossible for neural networks, which makes our experimental results questionable. But as stated in Section 3, all test cases of $k_1 \ln(k_2 x)$ are just one rule for Rosymtor. The abstraction of coefficients effectively solves the jitter problem, reducing its occurrence to almost negligible.

Table 4. Accuracy comparison on robustness test.

	LC model	Rosymtor
$k_1 \ln(k_2 x)$	100.0%	100.0%
$k_1 e^{k_2 x}$	24.5%	100.0%
$k_1 x^{k_2}$	19.3%	100.0%
$k_1 \sin(k_2 x)$	31.8%	100.0%
$k_1 \cos(k_2 x)$	31.8%	100.0%
$k_1 \tan(k_2 x)$	30.7%	100.0%

4.5 Evaluation of Compositionality

Afterward, we test the compositionality of the models. We randomly sample 1,000 composite expressions f_1, f_2, \dots, f_k , where $k \in 2, 3, 4$, and f_1, f_2, \dots, f_k are all primitive functions that the model can accurately integrate. These expressions form test sets, where we test the accuracy of LC model and Rosymtor, as shown in Table 5.

For LC model, while the accuracy of solving individual primitive functions is 100.00%, it decreases to 96.2% when solving two composite functions, 15.7% for three added together, and disappointingly drops to only 14.4% for four added together. By decomposing the expressions, Rosymtor achieves a remarkable 100% accuracy, demonstrating generalization in the aspect of compositionality. Note that all simple

function can be integrated correctly, and the integration is satisfied with the compositionality of addition. Therefore, 100% accuracy is reasonable.

Table 5. Accuracy comparison on compositionality test.

	LC model	Rosymtor
f_1	100.0%	100.0%
$f_1 + f_2$	96.2%	100.0%
$f_1 + f_2 + f_3$	15.7%	100.0%
$f_1 + f_2 + f_3 + f_4$	14.4%	100.0%

4.6 Evaluation of Out-of-distribution Generation

Finally, we test the out-of-distribution generalization, including larger values, longer problems, and functions not covered in training.

For larger values, we select the problems which both LC model and Rosymtor can originally integrate and then randomly scale up the coefficients by 10 times, 20 times, 50 times, and 100 times. We test whether the models can still predict these modified expressions accurately. LC model's accuracy sharply declines to 53% when the integers are changed by 50 times, while Rosymtor maintains a 100% accuracy throughout.

As to longer problems, we test accuracy on expressions with more operators, as shown in Table 6. Rosymtor shows a slightly stronger ability to handle longer problems, because additive decomposition allows solving parts separately, reducing the complexity of longer problems to some extent. However, not all long expressions can be decomposed by addition. Therefore, the accuracy of Rosymtor will also decrease as the number of operators increases.

Table 6. The impact of longer problems on models.

Operators	LC model	Rosymtor
1-15	97.4%	98.8%
20	89.1%	95.9%
25	81.7%	90.4%
30	59.3%	72.6%
35	46.3%	67.0%

We test learning ability for unknown functions using two trigonometric substitutions: $\tan = \sin / \cos$ and $\sin = \cos \times \tan$. Taking $\tan = \sin / \cos$ as an example, we select expressions from the dataset containing the function \tan and replace it with \sin / \cos so that the model is unaware of the existence of \tan . After obtaining the answers through the neural network, we check if they contain structures like \sin / \cos and replace them with \tan . The results of accuracy are shown in Table 7. When removing expressions containing \tan from the training set of LC model and

testing it on expressions containing \tan , LC model exhibits a significantly lower accuracy. In comparison, we find that substitution has little effect on accuracy, and our model displays generalization capabilities beyond the training data.

Table 7. The impact of unknown functions on models. (Beam size determines how many alternative sequences the beam search algorithm [19] keeps track of during the process of generating output.)

	Beam 1	Beam 5	Beam 10
$\tan = \sin / \cos$	72.1%	90.6%	97.8%
$\sin = \cos \times \tan$	69.9%	90.0%	98.3%

4.7 Experiment Summary

From the above experiments, it is easily observed that the combination of the sequence characteristic in machine translation and the mathematical characteristic in integration, can result in a more rational neural sequence integrator.

After replacing specific numbers, the neural network receives the same sequence with the abstraction of coefficients, thereby eliminating the influence of number changes. Decomposing expressions in advance makes the model adhere to the mathematical decomposition rules required for integration. Besides, by mathematical substitution, we avoid the troubles of expanding the vocabulary and achieve a cost-effective way to handle unknown words.

Although coefficient substitution and expression decomposition are not originally aimed at out-of-distribution generalization, they effectively improve the model's performance for larger values and longer problems. Coefficient substitution allows the model to process values based on their mathematical properties rather than common strings. Regardless of the magnitude of values, they can be extracted and replaced with coefficients that remain unaffected. On the other hand, expression decomposition may break longer expressions into shorter ones, which can reduce the difficulty and complexity of solving.

SAGGA is a genetic algorithm which can automatically discover diverse failures of neural sequence integrators [22]. We execute SAGGA on Rosymtor and get a result of no counterexample output after two hours, indicating that Rosymtor has passed the SAGGA test.

5 Conclusion

In this paper, we extend the work of [11] and propose a new integrator Rosymtor by enhancing sequence model with mathematical reasoning. Abstraction of coefficients makes Rosymtor more stable in minor perturbations. Moreover, expressions are first decomposed and then processed separately by the neural network, which enables Rosymtor to adhere to the compositional requirements of integration. Lastly, we

evaluate out-of-distribution generalization and achieve satisfactory experimental results. More specifically, Rosymtor also passes the SAGGA test. The success of Rosymtor means combining sequence model with mathematical reasoning can promote the performance of AI-based models and solve more complex problems.

Acknowledgments. This work is supported by NSFC (No. 62272416) and the National Key Research Project of China (No. 2023YFA1009402).

References

1. Agrawal, A., Batra, D., Parikh, D.: Analyzing the behavior of visual question answering models. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 1955-1960 (2016)
2. Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T.H., de Vries, H., Courville, A.: Systematic generalization: What is required and can it be learned? In: Proceedings of the 7th International Conference on Learning Representations (2019)
3. Chen, T., Chen, H.: Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Transactions on Neural Networks* 6(4), 904-910 (1995)
4. Davis, E.: The use of deep learning for symbolic integration: A review of (Lample and Charton, 2019). *arXiv preprint arXiv:1912.05752*. (2019)
5. Evans, R., Saxton, D., Amos, D., Kohli, P., Grefenstette, E.: Can Neural Networks Understand Logical Entailment. In: Proceedings of the 6th International Conference of Learning Representations (2018)
6. Feng, W., Liu, B., Xu, D., Zheng, Q., Xu, Y.: NeuReduce: Reducing mixed Boolean-arithmetic expressions by recurrent neural network. In: Proceedings of the Findings of the 2020 Empirical Methods in Natural Language Processing. pp. 635-644 (2020)
7. Hupkes, D., Dankers, V., Mul, M., Bruni, E.: Compositionality decomposed: How do neural networks generalize? In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. pp. 5065-5069 (2020)
8. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations (2015)
9. Lagaris, I., Likas, A., Fotiadis, D.: Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks* 9, 987-1000 (1998)
10. Lake, B., Baroni, M.: Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks. In: Proceedings of the 6th International Conference on Learning Representations (2018)
11. Lample, G., Charton, F.: Deep learning for symbolic mathematics. In: Proceedings of the 8th International Conference on Learning Representations (2020)
12. Lee, H., Kang, I.S.: Neural algorithm for solving differential equations. *Journal of Computational Physics* 91(1), 110-131 (1990)
13. Mitsuo, Takeda, Goodman, J.W.: Neural networks for computation: Number representations and programming complexity. *Applied Optics* 25(18), 3033-3046 (1986)
14. Nogueira, R.F., Jiang, Z., Lin, J.: Investigating the limitations of the transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*. (2021)
15. Noorbakhsh, K., Sulaiman, M., Sharifi, M., Roy, K., Jamshidi, P.: Pretrained language models are symbolic mathematics solvers too! *arXiv preprint arXiv:2110.03501*. (2021)

16. Piotrowski, B., Urban, J., Brown, C.E., Kaliszyk, C.: Can neural networks learn symbolic rewriting? In: Proceedings of the 9th International Conference on Learning Representations (2021)
17. Richardson, D.: Some undecidable problems involving elementary functions of a real variable. *The Journal of Symbolic Logic* 33(4), 514-520 (1968)
18. Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., Dill, D.L.: Learning a SAT solver from single-bit supervision. In: Proceedings of the 7th International Conference on Learning Representations (2019)
19. Tillmann, C., Ney, H.: Word Reordering and a Dynamic Programming Beam Search Algorithm for Statistical Machine Translation. *Computational Linguistics* 29(1), 97-133 (2003).
20. Trask, A., Hill, F., Reed, S.E., Rae, J., Dyer, C., Blunsom, P.: Neural arithmetic logic units. In: Advances in Neural Information Processing Systems, 8046-8055 (2018)
21. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Polosukhin, I.: Attention is all you need. In Advances in Neural Information Processing Systems 30, 6000-6010 (2017)
22. Welleck, S., West, P., Cao, J., Choi, Y.: Symbolic brittleness in sequence models: on systematic generalization in symbolic mathematics. In: Proceedings of the 36th AAAI Conference on Artificial Intelligence. pp. 8629–8637 (2022)