Neural Machine Translation for Mathematical Formulae

Felix Petersen[Σ], Moritz Schubotz[Π], André Greiner-Petter[∫], Bela Gipp[∫]

[Σ]Stanford University, [Π]FIZ Karlsruhe, [∫]University of Göttingen

mail@felix-petersen.de, moritz.schubotz@fiz-karlsruhe.de
greinerpetter@gipplab.com, gipp@uni-goettingen.de

Abstract

We tackle the problem of neural machine translation of mathematical formulae between ambiguous presentation languages and unambiguous content languages. Compared to neural machine translation on natural language, mathematical formulae have a much smaller vocabulary and much longer sequences of symbols, while their translation requires extreme precision to satisfy mathematical information needs. In this work, we perform the tasks of translating from LaTeX to Mathematica as well as from LaTeX to *semantic* LaTeX. While recurrent, recursive, and transformer networks struggle with preserving all contained information, we find that convolutional sequence-to-sequence networks achieve 95.1% and 90.7% exact matches, respectively.

1 Introduction

Mathematical notations consist of symbolic representations of mathematical concepts. For the purpose of displaying them, most mathematical formulae are denoted in presentation languages (PL) (Schubotz et al., 2018) such as LaTeX (Lamport, 1994). However, for computer-interpretation of formulae, machine-readable and unambiguous content languages (CL) such as Mathematica or *semantic* LaTeX are necessary. Thus, this work tackles the problem of neural machine translation between PLs and CLs as a crucial step toward machine-interpretation of mathematics found in academic and technical documents.

In the following, we will illustrate the ambiguities of representational languages. Those ambiguities range from a symbol having different meanings over notational conventions that change over time to a meaning having multiple symbols. Consider the ambiguous mathematical expression $(x)_n$. While Pochhammer (Pochhammer, 1870) himself used $(x)_n$ for the binomial coefficient $\binom{x}{n}$, for mathematicians in the subject area of special functions, more precisely hypergeometric series, $(x)_n$ usually denotes the Pochhammer symbol, which is

defined for natural numbers as

$$(x)_n := x^{\overline{n}} = \prod_{k=0}^{n-1} (x + k). \tag{1}$$

To further complicate matters, in statistics and combinatorics, the same notation is defined as

$$(x)_n := x^{\underline{n}} = \prod_{k=0}^{n-1} (x - k). \tag{2}$$

This work uses LaTeX as PL and Mathematica as well as *semantic* LaTeX as CLs. Mathematica is one of the most popular Computer Algebra Systems (CASs); we use Mathematica's standard notation (InputForm) as a CL (from now on, for simplicity, referred to as Mathematica.) *Semantic* LaTeX is a set of LaTeX macros that allow an unambiguous mathematical notation within LaTeX (Miller and Youssef, 2003) and which has been developed at the National Institute for Standards and Technology (NIST) by the Digital Library of Mathematical Functions (DLMF) and the Digital Repository of Mathematical Formulae (DRMF).

In LaTeX, the Pochhammer symbol $(x)_n$ is simply denoted as `(x)_n`. In *semantic* LaTeX, it is denoted as `\Pochhammersym{x}{n}` and compiled to LaTeX as `{\left(x\right)_{n}}`. In Mathematica, it is denoted as `Pochhammer[x, n]` and can be exported to LaTeX as `(x)_n`.

To display them, it is generally possible to translate formulae from CLs to PLs, e.g., Mathematica has the functionality to export to LaTeX, and *semantic* LaTeX is translated into LaTeX as a step of compilation. However, the reverse translation from PL to CL is ambiguous because semantic information is lost when translating into a PL.

Mathematical formulae are generally similar to natural language (Greiner-Petter et al., 2020). However, mathematical formulae are often much longer than natural language sentences. As an example of sentence lengths, 98% of the sentences in the Stanford Natural Language Inference entailment task contain less than 25 words (Bowman et al.,
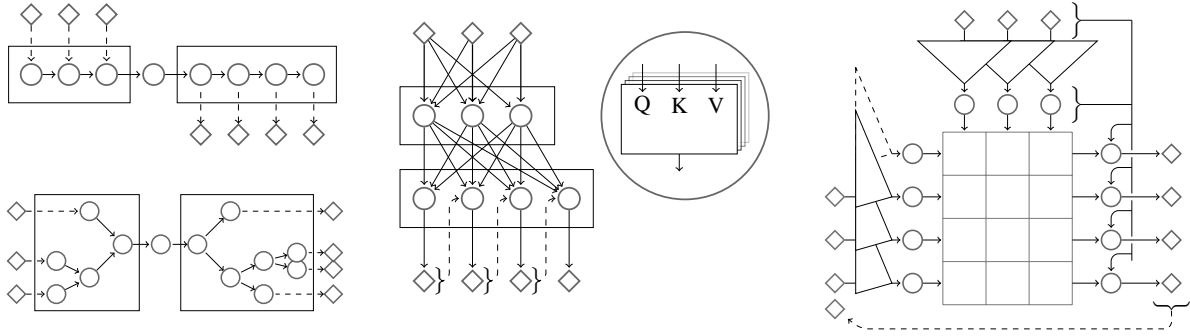
1

Figure 1: Schema of recurrent (top left), recursive (bottom left), transformer (middle), and convolutional sequence-to-sequence (right) neural networks.

2016). In contrast, the average number of Mathematica tokens in the Mathematical Functions Site data set is 173, only 2.25% of the formulae contain less than 25 tokens, and 2.1% of the formulae are longer than 1 024 tokens. At the same time, mathematical languages commonly require only small vocabularies of around 1 000 tokens (relative to natural languages.)

By applying convolutional sequence-to-sequence networks, this work achieves an exact match accuracy of 95.1% for a translation from LATEX to Mathematica as well as an accuracy of 90.7% for a translation from LATEX to *semantic* LATEX. In contrast, the import function of the Mathematica software achieves an exact match accuracy of 2.7%. On all measured metrics, our model outperforms export / import round trips using Mathematica.

2 Related Work

2.1 Neural Machine Translation

The most common neural machine translation models are sequence-to-sequence recurrent neural networks (Sutskever et al., 2014), tree-structured recursive neural networks (Goller and Kuchler, 1996), transformer sequence-to-sequence networks (Vaswani et al., 2017), and convolutional sequence-to-sequence networks (Gehring et al., 2017). In the following, we sketch the core principle of these network types, which are displayed in Figure 1.

Recurrent sequence-to-sequence neural networks (Figure 1, top left) are networks that process the tokens one after each other in a linear fashion. Note that the longest shortest path in this architecture is the sum of the length of the input and the length of the output. An attention mechanism can reduce the loss of information in the network (not shown in the schema).

Recursive tree-to-tree neural networks (Figure 1, bottom left) are networks that process the input in a tree-like fashion. Here, the longest shortest path is the sum of the depths of input and output, i.e., logarithmic in the number of tokens.

Transformer sequence-to-sequence neural networks (Figure 1, middle) allow a dictionary-like lookup of hidden states produced from the input sequence. This is possible through an elaborate multi-headed attention mechanism.

Convolutional sequence-to-sequence neural networks (Figure 1, right) process the input using a convolutional neural network and use an attention-mechanism to attribute which input is most relevant for predicting the next token given previously predicted tokens.

In natural language translation, transformer networks perform best, convolutional second best, and recurrent third best (Gehring et al., 2017; Vaswani et al., 2017; Ott et al., 2018). Recursive neural networks are commonly not applicable to natural language translation.

2.2 Rule-Based Formula Translation

LATEXML is a LATEX to XML Converter (Ginev and Miller, 2013). It can translate from *semantic* LATEX to LATEX. As semantic information is lost during this process, a rule-based back-translation is not possible.

Mathematica can export expressions into LATEX and also import from LATEX. However, the import from LATEX uses strict and non-exhaustive rules that oftentimes do not translate into the original Mathematica expressions, e.g., we found that only 3.1% of expressions exported from Mathematica to LATEX and (without throwing an error) imported back into Mathematica are exact matches. This is because, when translating into LATEX, the semantic

information is lost. Moreover, we found that 11.5% of the formulae exported from Mathematica throw an error when reimporting them.

For the translation between CLs, from *semantic* LaTeX to CASs and back, there exists a rule-based translator (Cohl et al., 2017; Greiner-Petter et al., 2019). The *semantic* LaTeX to Maple translator achieved an accuracy of 53.59% on correctly translating $4\,165$ test equations from the DLMF (Greiner-Petter et al., 2019). The accuracy of the *semantic* LaTeX to CAS translator is relatively low due to the high complexity of the tested equations and because many of the functions which are represented by a DLMF/DRMF LaTeX macro are not defined or defined differently in Maple (Greiner-Petter et al., 2019).

2.3 Deep Learning for Mathematics

Lample and Charton (2020) used deep learning to solve symbolic mathematics problems. They used a sequence-to-sequence transformer model to translate representations of mathematical expressions into representations of solutions to problems such as differentiation or integration. In their results, they outperform CASs such as Mathematica.

Wang et al. (2018) used a recurrent neural network-based sequence-to-sequence model to translate from LaTeX (text including formulae) to the Mizar language, a formal language for writing mathematical definitions and proofs. Their system generates correct Mizar statements for 65.7% of their synthetic data set.

Other previous works (Deng et al., 2017; Wang et al., 2019) concentrated on the "image2latex" task, which was originally proposed by OpenAI. This task's concept is the conversion of mathematical formulae in images into LaTeX, i.e., optical character recognition of mathematical formulae. Deng et al. (2017) provide im2latex-100k, a data set consisting of about $100\,000$ formulae from papers of arXiv, including their renderings. They achieved an accuracy of 75% on synthetically rendered formulae. Compared to the data sets used in this work, the formulae in im2latex-100k are much shorter.

This was followed by other relevant lines of work by Wu et al. (2021); Zhang et al. (2020); Li et al. (2022); Ferreira et al. (2022); Patel et al. (2021).

Table 1: Data set summary statistics. Format for number of characters per formula/format: Mean±Std. (Median).

Data Set	Formulae	Input (LaTeX)	Output (Mat. / *sem.* L.)
Mathematica	$307\,672$	345.5 ± 534.4 (195)	320.7 ± 585.7 (168)
semantic LaTeX	$11\,639$	163.8 ± 246.2 (116)	145.6 ± 230.1 (103)

3 Training Data Sets & Preprocessing

Mathematical Functions Site Data Set. The "Mathematical Functions Site"[1] by Wolfram Research is a repository of $307\,672$ mathematical formulae available in Mathematica InputForm format. By web-crawling, we retrieved all formulae in Mathematica InputForm and (using Mathematica v12.0) exported the formulae from Mathematica into LaTeX.

***Semantic* LaTeX Data Set.** The *semantic* LaTeX data set consist of $11\,639$ pairs of formulae in the LaTeX and *semantic* LaTeX formats generated by translating from *semantic* LaTeX to LaTeX using LaTeXML. Cohl et al. (2015) provided us this unreleased data set.

Preprocessing We preprocessed the data sets by tokenizing them with custom rule-based tokenizers for LaTeX and Mathematica. Note that as *semantic* LaTeX follows the rules of LaTeX, we can use the same for both cases. Details on the tokenizers are presented in the supplementary material. For recursive neural networks, we parsed the data into respective binary trees in postfix notation.

We randomly split the Mathematical Functions Site data set into disjoint sets of 97% training, 0.5% validation, and 2.5% test data and split the *semantic* LaTeX data set into 90% training, 5% validation, and 5% test data since this data set is smaller. Data set summary statistics can be found in Table 1.

4 Methods

We briefly discuss recurrent, recursive, and transformer architectures and then discuss convolutional sequence-to-sequence networks in detail because they showed, by far, the best results.

Recurrent Neural Networks showed the worst performance. Our experiments used Long-Short-Term-Memory (LSTM) recurrent networks but did not achieve any exact matches on long equations of the *semantic* LaTeX data set. This is not surprising as recurrent neural networks generally have

---

[1] http://functions.wolfram.com/

poor performance regarding long-term relationships spanning over hundreds of tokens (Trinh et al., 2018). For our data sets, the longest shortest path in the neural network easily exceeds $2\,000$ blocks. Note that the exact match accuracy on such long equations produces successful responses only for a very well-performing model; getting most symbols correct does not constitute an exact match. For a definition of exact matches, see Section 5.1.

Recursive Neural Networks showed slightly better performance of up to 4.4% exact match accuracy when translating from LaTeX into *semantic* LaTeX. This can be attributed to the fact that the longest path inside a recursive neural network is significantly shorter than in a recurrent neural network (as the longest shortest path in a tree can be much shorter than the longest shortest path in a sequence.) Further, an additional traversal into postfix notation allows for an omission of most braces/parentheses, which (on the *semantic* LaTeX data set) reduced the required amount of tokens per formula by about $20 - 40\%$. Similar to the recurrent networks, we also used LSTMs for the recursive networks. Note that training recursive neural networks is hard because they cannot easily be batched if the topology of the trees differs from sample to sample, which it does for equations.

Transformer Neural Networks significantly outperform previous architectures. In our best experiments, we achieved performances of up to 50% exact matches on the Mathematical Functions Site data set. This leap in performance can be attributed to the elaborate multi-headed attention mechanism underlying the transformer model. Because we experimented simultaneously with the convolutional sequence-to-sequence architecture and the transformer architecture, and the performance of convolutional networks was by a large margin better ($> 90\%$) than the best performance on transformer neural networks, we decided to set the focus of this work on convolutional networks only. We note that in natural language translation, transformer models typically outperform convolutional neural networks (Gehring et al., 2017; Vaswani et al., 2017; Ott et al., 2018).

4.1 Convolutional Seq-to-Seq Networks

In contrast to recurrent and recursive neural networks, convolutional sequence-to-sequence networks do not need to compress the relevant information. Due to the attention matrix architecture,

the convolutional model can easily replicate the identity, a task that recurrent and recursive neural networks struggle with. In fact, an above 99% accuracy can be achieved on learning the identity within the first epoch of training. Given that the syntax of two languages follows the same paradigm, the translation is often not far from the identity, e.g., it is possible that only some of the tokens have to be modified while many remain the same. This separates mathematical notations from natural languages.

In the following, we discuss hyperparameters and additional design choices for convolutional networks. Note that the models for each language pair are independent. In Supplementary Material C, we provide respective ablation studies.

Learning Rate, Gradient Clipping, Dropout, and Loss. Following the default for this model, we use a learning rate of 0.25, applied gradient clipping on gradients greater than 0.1, and used a dropout rate of 0.2. As a loss, we use label-smoothed cross-entropy.

State/Embedding Size(s). We found that a state size of 512 performs best. In this architecture, it is possible to use multiple state sizes by additional fully connected layers between convolutional layers of varying state size. In contrast to the convolutional layers, fully connected layers are not residual and thus increase the length of the shortest path in the network. We found that networks with a single state size performed best. Note that while in natural language translation, with vocabularies of $40\,000 - 200\,000$ tokens, a state size of 512 is also commonly used (Gehring et al., 2017), while our examined mathematical languages contain only $500 - 1\,000$ tokens. That a state size of 256 performed significantly worse (88.3% for 256 and 94.9% for 512) indicates a high entropy/information content of the equations.

Number of Layers. We found that 11 layers perform best.

Batch Size. We found that $48\,000$ tokens per batch perform best. This is equivalent to a batch size of about 400 formulae.

Kernel Size. We use a kernel size of 3. We found that a kernel size of 5 performs by 0.1% better than a kernel size of 3, but as the larger kernel size also requires much more parameters and is more expensive to compute, we decided to go with 3.

Substitution of Numbers. Since the Mathematical Functions Site data set contains more than 10^4 multi-digit numbers, while it contains less than 10^3 non-numerical tags, these numbers cannot be interpreted as conventional tags. Thus, numbers are either split into single digits or replaced by variable tags. Splitting numbers into single digits causes significantly longer token streams, which degrades performance. Substituting all multi-digit numbers with tags like `<number_01>` improved the exact match accuracy of the validation data set from 92.7% to 95.0%. We use a total of 32 of such placeholder tags as more than 99% of the formulae have less or equal to 32 multi-digit numbers. We randomly select the tags that we substitute the numbers with. Since multi-digit numbers basically always perfectly correspond in the different mathematical languages, we directly replace the tag with their corresponding numbers after the translation.

LightConv. As an alternative to the model proposed by Gehring et al. (2017), we also used the LightConv model as presented by Wu et al. (2019). As expected, this model did not yield good results on mathematical formula translation as it does not use the strong self-attention that the model by Gehring et al. (2017) has. Note that LightConv outperforms the convolutional sequence-to-sequence model by Gehring et al. (2017) on natural language (Wu et al., 2019).

5 Evaluation of the Convolutional Network

5.1 Evaluation Metrics

Exact Match (EM) Accuracy The EM accuracy is the non-weighted share of exact matches. An exact match is defined as a translation of a formula where every token equals the ground truth. This makes the EM accuracy an extremely strict metric as well as a universal and definite statement about a lower bound of the quality of the translation. For example, the exact match might fail since $E = mc^2$ can be written as both `E=mc^2` and `E=mc^{2}`, which is, although content-wise equal, not an exact match. However, in our experiments, such errors do not occur regularly since, for the generation of the synthetic training data, the translation was performed using the rule-based translators Mathematica and LaTeXML. Only 0.4% of the erroneous translations to *semantic* LaTeX were caused by braces (`{`, `}`). In none of these cases the

braces were balanced, i.e., each of these formulae was semantically incorrect. For the translation to Mathematica, only 0.02% of the formulae did not achieve an exact match due to brackets (`[`, `]`).

Levenshtein Distance (LD) The LD, which is also referred to as "edit distance", is the minimum number of edits required to change one token stream into another (Levenshtein, 1966). This metric reflects the error in a more differentiated way. We denote the share of translations that have a Levenshtein distance of up to 5 by $\mathrm{LD}_{\leq 5}$ and denote the average Levenshtein Distance by LD.

Bilingual Evaluation Understudy (BLEU) The BLEU score is a quality measure that compares the machine's output to a translation by a professional human translator (Papineni et al., 2002). It compares the n-grams (specifically $n \in \{1, 2, 3, 4\}$) between the prediction and the ground truth. Since the translations in the data sets are ground truth values instead of human translations, for the back-translation of formulae, this metric reflects the closeness to the ground truth. BLEU scores range from 0 to 100, with a higher value indicating a better result. For natural language on the WMT data set, state-of-the-art BLEU scores are 35.0 for a translation from English to German and 45.6 for a translation from English to French (Edunov et al., 2018). That the BLEU scores for formula translations are significantly higher than the scores for natural language can be attributed to the larger vocabularies in natural language and a considerably higher variability between correct translations. In contrast, in most cases of formula translation, the translation is not ambiguous. We report the BLEU scores to demonstrate how BLEU scores behave on strictly defined languages like mathematical formulae.

Perplexity The perplexity is a measurement of how certain a probability distribution is to predict a sample. Specifically, the perplexity of a discrete probability distribution p is generally defined as

$$\mathtt{ppl}(p) = 2^{H(p)} = 2^{-\sum_x p(x)\log_2 p(x)} \quad (3)$$

where H denotes the entropy, and x is drawn from the set of all possible translations (Cover and Thomas, 2006). In natural language processing, a lower perplexity indicates a better model. As we will discuss later, this does not hold for mathematical language.

5.1.1 Discussion on the Perplexity of Mathematical Language Translations

In natural language translation, the perplexity is a common measure for selecting the epoch at which the performance on the validation set is best. That is because its formulation is very similar to the employed cross-entropy loss. This procedure avoids overfitting and helps to select the best-performing epoch without having to compute the actual translations. Computing the translations would be computationally much more expensive because it requires a beam search algorithm, and the quality of a resulting translation cannot be measured by a simple metric such as EM.

However, for formula translation, the perplexity does not reflect the accuracy of the model. While the validation accuracy rises over the course of the training, the rising perplexity falsely indicates that the model's performance decays during training. We presume that this is because the perplexity reflects how sure the model is about the prediction instead of whether the prediction with the highest probability is correct. Since many subexpressions of mathematical formulae (e.g., `n + 1`) are invariant to translations between many mathematical languages, the translations are closer to the identity than translations between natural languages. Therefore, a representation very close to the identity is learned first. Consecutively, this translation is transformed into the actual translation. Empirically, the validation perplexity usually reaches its minimum during the first epoch. Afterward, when the translation improves, the uncertainty (perplexity) of the model also increases. Thus, we do not use the perplexity for early stopping but instead compute the EM accuracy on the validation set.

5.2 Evaluation Techniques

Back-Translation. As, for the training data sets, only the content language (i.e., Mathematica / *semantic* LaTeX, respectively) was available, we programmatically generated the input forms (presentation language) using Mathematica's conversion and the LaTeX macro definitions of *semantic* LaTeX, respectively. This process corresponds to the internal process for displaying Mathematica / *semantic* LaTeX equations in LaTeX form. Thus, the task is to back-translate from (ambiguous) LaTeX to the (unambiguous) Mathematica / *semantic* LaTeX forms.

Additional Experiments. In addition to this, we also perform round trip experiments from LaTeX

Table 2: Main results for the back-translation.

Metric	LaTeX → Mathematica	LaTeX → *semantic* LaTeX
EM	95.1%	90.7%
BLEU	99.68	96.79

Table 3: Comparison between Mathematica and our model on back-translating the formulae of the Mathematical Functions Site data set. Import denotes the fraction of formulae that can be imported by Mathematica, i.e., whether Mathematica can import the LaTeX format or whether our model produces valid Mathematica syntax, respectively.

Method	EM	Import	$LD_{\leq 5}$	LD
Mathematica	2.7%	88.5%	16.4%	88.7
Conv. Seq2Seq	**95.1%**	**98.3%**	**96.7%**	**0.615**

into Mathematica and back again on the im2latex-100k data set. Here, we use our model as well as the Mathematica software to translate from LaTeX into Mathematica. In both cases, we use Mathematica to translate back into LaTeX. The im2latex-100k data set contains equations as well as anything else that was typeset in math environments LaTeX. 66.8% of the equations in the im2latex-100k data set contain tokens that are not in the vocabulary. We note that an exact match is only possible if a LaTeX expression coincides with what would be exported from Mathematica. Thus, we did not expect large accuracy values for this data set.

5.3 Evaluation Results

Back-Translation. For the back-translation from LaTeX to Mathematica, we achieved an EM accuracy of 95.1% and a BLEU score of 99.68. That is, 95.1% of the expressions from Mathematica, translated by Mathematica into LaTeX can be translated back into Mathematica by our model without changes. For the translation from LaTeX to *semantic* LaTeX, we achieved an EM accuracy of 90.7% and a BLEU score of 96.79. The translation from LaTeX to *semantic* LaTeX performs not as well as the translation to Mathematica, i.a., because the *semantic* LaTeX data set is substantially smaller than the Mathematical Functions Site data set. The low LaTeX to *semantic* LaTeX BLEU score of only 96.79 is because the translations into *semantic* LaTeX are on average 2% shorter than the ground truth references. Note that 96.0% of the translations to *semantic* LaTeX had an LD of up to 3. The results are displayed in Table 2.

For comparing our model to the LaTeX import func-

tion of Mathematica, we show the results in Table 3. The low performance of Mathematica's LaTeX importer can be attributed to the fact that Symbols with a defined content/meaning, e.g., `DiracDelta` are exported to LaTeX as `\delta`, i.e., just as the character they are presented by. Since `\delta` is ambiguous, Mathematica interprets it as `\[Delta]`. With neural machine translation, on the other hand, the meaning is inferred from the context and, thus, it is properly interpreted as `DiracDelta`.

Additional Experiments. As for the round trip experiments, Mathematica was able to import 15.3% of the expressions in the im2latex-100k data set, while our model was able to generate valid Mathematica syntax for 16.3% of those expressions. For the im2latex-100k data set, the round trip experiment is ill-posed since the export to LaTeX will only achieve an exact match if the original LaTeX equation is written in the style in which Mathematica exports. However, as the same Mathematica export function is used for testing for exact matches, neither our model nor the Mathematica software has an advantage on this problem, which allows for a direct comparison. Mathematica achieved an exact match round trip in 0.153% and our model in 0.698% of the equations. The average LD for Mathematica is 18.3, whereas it is 12.9 for our model. We also note that while im2latex-100k primarily contains standard equations, our model is specifically trained to interpret equations with special functions. The results are presented in Table 4.

5.4 Qualitative Analysis

We present a qualitative analysis of the back-translations from LaTeX to Mathematica with the help of randomly selected positive and negative examples. The referenced translations / equations are in the supplementary material. All mentioned parts of equations will be marked in bold in the supplementary material. We want to give a small qualitative analysis of the translation from LaTeX to Mathematica and show in which cases the translation can fail, and give an intuition about why issues arise in these cases. In the supplementary material, further qualitative analysis is provided.

In Equation B.1, $\sigma_k(n)$ is correctly interpreted by our model as a `DivisorSigma`. Mathematica interprets it as the symbol σ with the subscript k, i.e., the respective semantic information is lost. At the end of this formula, the symbol $\wedge$ (`\land`) is properly interpreted by our model as `&&`. In

Table 4: Round trip experiment with the im2latex-100k (Deng et al., 2017) LaTeX expressions. Import denotes the fraction of formulae that can be imported by Mathematica, i.e., whether Mathematica can import the LaTeX format or whether our model produces valid Mathematica syntax, respectively. In this experiment, exact matches can only occur coincidental, i.e., a perfect translation by the model does not necessarily produce an exact match.

Method	EM	Import	LD$_{\leq 5}$	LD
Mathematica	0.153%	15.3%	2.30%	18.3
Conv. Seq2Seq	$\mathbf{0.698\%}$	$\mathbf{16.3\%}$	$\mathbf{2.56\%}$	$\mathbf{12.9}$

contrast, Mathematica interpreted it as `\[Wedge]`, which corresponds to the same presentation but without the underlying definition that is attached to `&&`. In this equation, our approach omitted one closing bracket at a place where two consecutive closing brackets should have been placed.

In Equation B.2, the symbol $\wp$ (`\wp`) is properly interpreted by the model and Mathematica as the Weierstrass' elliptic function $\wp$ (`WeierstrassP`). That is because the symbol $\wp$ is unique to the Weierstrass $\wp$ function. The inverse of this function, $\wp^{-1}$ is also properly interpreted by both systems as the `InverseWeierstrassP`. Our model correctly interprets the sigmas in the same equation as the `WeierstrassSigma`. As σ does not have a unique meaning, Mathematica just interprets it as a bare sigma `\[Sigma]`. The difference between our translation and the ground truth is that our translation omitted a redundant pair of parentheses.

Equation B.3 displays an example of the token `<number_XX>`, which operates as a replacement for multi-digit numbers. In this example, our model interprets $Q_4^9(z)$ as `GammaRegularized[4, 9, z]` instead of the ground truth `LegendreQ[4, 9, 3, z]`. This case is especially hard since the argument "3" is not displayed in the LaTeX equation and `LegendreQ` has commonly only two to three arguments.

Equation B.7 is correctly interpreted by our model including the expression `\int \sin (az) ... dz`. Note that Mathematica fails at interpreting `\int 2z dz`[2].

To test whether our model can perform translations on a data set that was generated by a different engine, we perform a manual evaluation on

---

[2]The command `ToExpression["\\int 2z dz", TeXForm, Defer]` fails for Mathematica v. 12.0

translations from LaTeX to Mathematica for the DLMF data set (generated by LaTeXML). To test our model, which was trained on LaTeX expressions produced by Mathematica, on LaTeX expressions produced by LaTeXML, we used a data set of 100 randomly selected expressions from the DLMF, which is written in *semantic* LaTeX. A caveat of this is that LaTeXML produces a specific LaTeX flavor in which some mathematical expressions are denoted in an unconventional fashion[3]. As 71 of those 100 expressions contain tokens that are not in the Mathematica-export vocabulary, these cannot be interpreted by the model. Further, as LaTeX is very flexible, a large variety of LaTeX expressions can produce a visually equivalent result; even among a restricted vocabulary, there are many equivalent LaTeX expressions. This causes a significant distributional domain shift between LaTeX expressions generated by different systems. Our model generates valid and semantically correct Mathematica representations for 5 equations. Specifically, in equations (4.4.17), (8.4.13), and (8.6.7), the model was able to correctly anticipate the incomplete Gamma function and Euler's number e.

This translation from DLMF to Mathematica is difficult for several reasons as explained by Greiner-Petter et al. (2019). In their work, they translate the same 100 equations, however, from *semantic* LaTeX into Mathematica, using their rule-based translator, which was designed for this specific task (Greiner-Petter et al., 2019). On this different task, they achieved an accuracy of only 56%, which clearly shows how difficult a translation between two systems is even when the semantic information is explicitly provided by *semantic* LaTeX expressions.

In comparison, when the vocabulary of LaTeXML and Mathematica intersects, our model achieves a 17% accuracy while only inferring the implicit semantic information (i.e., the semantic information that can be derived from the structure of and context within a LaTeX expression).

6 Limitations

In this work, we evaluated neural networks on the task of back-translating mathematical formulae from the PL LaTeX to semantic CLs. For this purpose, we explored various types of neural networks and found that convolutional neural networks perform best. Moreover, we observed that the perplexity of the translation of mathematical formulae behaves differently from the perplexity of the translation between natural languages.

Our evaluation shows that our model outperforms the Mathematica software on the task of interpreting LaTeX produced by Mathematica while inferring the semantic information from the context within the formula.

A general limitation of neural networks is that trained models inherit biases from training data. For a successful formula translation, this means that the set of symbols, as well as the style in which the formulae are written, has to be present in the training data. Mathematica exports into a very common flavor / convention of LaTeX, while *semantic* LaTeX, translated by LaTeXML, yields many unconventional LaTeX expressions. In both cases, however, the flavor / conventions of LaTeX are constant and do not allow variation as it is produced by a rule-based translator. Because of the limited vocabularies as well as limited set of LaTeX conventions in the data sets, the translation of mathematical LaTeX expressions of different flavors is not possible. In addition, we can see that a shift to a more difficult domain, such as special functions in the DLMF, produces a drop in performance but still generates very promising results. In future work, the translator could be improved by augmenting the data set such that it uses more and different ways to express the same content in the source language. As an example, a random choice between multiple ways to express a Mathematica expression in LaTeX could be added. For *semantic* LaTeX, the performance on real-world data could be improved by using multiple macro definitions for each macro. Ideal would be a data set of hand-written equivalents between the PLs and CLs. An addition could be multilingual translation (Johnson et al., 2017; Blackwood et al., 2018). This could allow learning translations and tokens that are not present in the training data for the respective language pair. Further, mathematical language-independent concepts could support a shared internal representation.

Another limitation is that data sets of mathematical formulae are not publicly available due to copyright and licensing. We will attempt to mitigate this issue by providing the data sets to interested researchers.

Note that this work does not use information from the context around a formula. Integrating

---

such context information would aid the translation as it can solve ambiguities. For example, for interpreting the expression $(x)_n$, information about the specific field of mathematics is essential. Further, context information can include custom mathematical definitions. In real-world applications, building on such additional information could be important for reliable translations.

7 Conclusion

In this work, we have shown that neural networks, specifically convolutional sequence-to-sequence networks, can handle even long mathematical formulae with high precision. Given an appropriate data set, we believe that it is possible to train a reliable formula translation system for real-world applications.

We hope to inspire the research community to apply convolutional neural networks rather than transformer networks to tasks that operate on mathematical representations (Deng et al., 2017; Matsuzaki et al., 2017; Lample and Charton, 2020; Wang et al., 2018; Wu et al., 2021; Zhang et al., 2020; Patel et al., 2021; Li et al., 2022; Ferreira et al., 2022). We think that convolutional networks could also improve program-to-program translation as source code has strong similarities to digital mathematical notations—after all, LATEX and Mathematica are programming languages.

Acknowledgments

References

Graeme Blackwood, Miguel Ballesteros, and Todd Ward. 2018. Multilingual neural machine translation with task-specific attention. In *Proceedings of the 27th International Conference on Computational Linguistics*.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Howard S Cohl, Moritz Schubotz, Marjorie A McClain, Bonita V Saunders, Cherry Y Zou, Azeem S Mohammed, and Alex A Danoff. 2015. Growing the Digital Repository of Mathematical Formulae with Generic LATEX Sources. In *Intelligent Computer Mathematics (CICM)*.

Howard S. Cohl, Moritz Schubotz, Abdou Youssef, André Greiner-Petter, Jürgen Gerhard, Bonita V. Saunders, Marjorie A. McClain, Joon Bang, and Kevin Chen. 2017. Semantic preserving bijective mappings of mathematical formulae between document preparation systems and computer algebra systems. In *Intelligent Computer Mathematics (CICM)*.

Thomas M Cover and Joy A Thomas. 2006. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *International Conference on Machine Learning (ICML)*.

Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M. Rush. 2017. Image-to-markup generation with coarse-to-fine attention. In *International Conference on Machine Learning (ICML)*.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500. Association for Computational Linguistics.

Deborah Ferreira, Mokanarangan Thayaparan, Marco Valentino, Julia Rozanova, and Andre Freitas. 2022. To be or not to be an integer? encoding variables for mathematical text. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 938–948, Dublin, Ireland. Association for Computational Linguistics.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *International Conference on Machine Learning (ICML)*.

Deyan Ginev and Bruce R. Miller. 2013. Latexml 2012 - a year of latexml. In *Intelligent Computer Mathematics (CICM)*.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*.

André Greiner-Petter, Moritz Schubotz, Fabian Müller, Corinna Breitinger, Howard S. Cohl, Akiko Aizawa, and Bela Gipp. 2020. Discovering mathematical objects of interest - A study of mathematical notations. In *WWW: The Web Conference*.

André Greiner-Petter, Moritz Schubotz, Howard S. Cohl, and Bela Gipp. 2019. Semantic preserving bijective mappings for expressions involving special functions between computer algebra systems and document preparation systems. *Aslib Journal of Information Management*.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *Transactions of the Association for Computational Linguistics*.

Guillaume Lample and François Charton. 2020. Deep learning for symbolic mathematics.

Leslie Lamport. 1994. *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley.

V. Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*.

Zhongli Li, Wenxuan Zhang, Chao Yan, Qingyu Zhou, Chao Li, Hongzhi Liu, and Yunbo Cao. 2022. Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2486–2496, Dublin, Ireland. Association for Computational Linguistics.

Takuya Matsuzaki, Takumi Ito, Hidenao Iwane, Hirokazu Anai, and Noriko H. Arai. 2017. Semantic parsing of pre-university math problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 2131–2141. Association for Computational Linguistics.

Bruce Miller and Abdou Youssef. 2003. Technical Aspects of the Digital Library of Mathematical Functions. *Annals of Mathematics and Artificial Intelligence*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Proc. Neural Information Processing Systems (NeurIPS)*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *NAACL 2021*.

Leo Pochhammer. 1870. Ueber hypergeometrische Functionen nter Ordnung. *Journal für die reine und angewandte Mathematik (Crelles Journal)*.

Moritz Schubotz, André Greiner-Petter, Philipp Scharpf, Norman Meuschke, Howard S Cohl, and Bela Gipp. 2018. Improving the Representation and Conversion of Mathematical Formulae by Considering their Textual Context. In *ACM/IEEE Joint Conference on Digital Libraries (JCDL)*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks.

Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. 2018. Learning longer-term dependencies in rnns with auxiliary losses. In *International Conference on Machine Learning (ICML)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. Neural Information Processing Systems (NeurIPS)*.

Jian Wang, Yunchuan Sun, and Shenling Wang. 2019. Image to latex with densenet encoder and joint attention.

Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. 2018. First Experiments with Neural Translation of Informal to Formal Mathematics.

Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. 2019. Pay Less Attention with Lightweight and Dynamic Convolutions. In *International Conference on Learning Representations (ICLR)*.

Qinzhuo Wu, Qi Zhang, Zhongyu Wei, and Xuanjing Huang. 2021. Math word problem solving with explicit numerical values. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 5859–5869. Association for Computational Linguistics.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020. Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 3928–3937. Association for Computational Linguistics.

A Implementation Details

For the implementation of the transformer and convolutional sequence-to-sequence models, we built on the PyTorch (Paszke et al., 2019) library fairseq (Ott et al., 2019) by Facebook AI Research. Since, as depicted in the evaluation, the perplexity does not properly reflect the accuracy, we extended fairseq by an option to measure the quality of models with an exact match accuracy and the Levenshtein distance.

We performed the computations on GPU-accelerated hardware. For the experiments, we used a server with 8 Nvidia Tesla V100 GPUs. Training took, depending on the setup, $1 - 96$ hours.

For performance, memory, and architectural reasons, for training, we only consider formulae with up to 1 024 tokens following the trend of current research (Bowman et al., 2016; Dauphin et al., 2017).

A.1 Tokenizers

For LaTeX, we developed a tokenizer for which LaTeX commands, parentheses, braces, brackets, as well as special characters are individual tokens. Letters are considered individual tokens and are thus split into single letters. Multi-digit numbers are considered as tokens as described in Section 4.1. For **Mathematica**, we developed a tokenizer that considers Mathematica functions, Symbols (e.g., \[Zeta]), parentheses, braces, brackets, as well as special characters are individual tokens.

As most strings of letters are Mathematica functions, we also consider all strings of letters as individual tokens, i.e., we do not split them into single letters. Multi-digit numbers are considered as tokens as described in Section 4.1. In addition, the following are exceptional tokens: &&, ==, <=, >=, !=, and /;.

For examples, see Supplementary Material B, which contains equations tokenized by our tokenizers.

B Qualitative Analysis – Mathematical Functions Site

This supplementary material presents 8 translation samples from LaTeX to Mathematica. All samples are randomly selected, only restricted by the editorial constraint of fitting on a single page, and restricted to have a similar amount of exact matches and erroneous cases. Equations B.1–B.4 are erroneous translations, while Equations B.5–B.8 lead to exact matches.

Here, the LaTeX formulae were generated by Mathematica's export function.

Table 5: **[1/3]** Equations including the tokenized LATEX input, the (optional) interpretation by Mathematica (Mat.), our translation (NMT), and the ground truth (GT.). (NMT+GT.) denotes that our translation is equal to the ground truth.

$$\sigma_k(n) = \prod_{j=1}^{m} \sigma_k\left(p_j^{n_j}\right) /; \text{FactorInteger}[n] = \{\{p_1, n_1\}, \ldots, \{p_m, n_m\}\} \land p_j \in \mathbb{P} \land k \in \mathbb{Z} \land n \in \mathbb{Z} \land n > 0 \qquad \text{(B.1)}$$

LATEX	`\sigma _ k (n) = \prod _ { j = 1 } ^ m \sigma _ k \left(p _ j ^ { n _ j } \right) \text{/;} \text{FactorInteger} [n] = \le f t \{ \le f t \{ p _ 1 , n _ 1 \right \} , \ldots , \le f t \{ p _ m , n _ m \right \} \right \} \land p _ j \in \mathbb{P} \land k \in \mathbb{Z} \land n \in \mathbb{Z} \land n > 0`
Mat.	`Subscript[\[Sigma], k][n] == \!\(*UnderoverscriptBox[\(\[Product]\), \(j = 1\), \(m\)]\(\(\(*SubscriptBox[\(\[Sigma]\), \(k\)]\)[*SubsuperscriptBox[\(p\), \(j\), SubscriptBox[\(n\), \(j\)]]]\)\) /; FactorInteger[n] == {{Subscript[p, 1], Subscript[n, 1]}, \[Ellipsis], {Subscript[p, m], Subscript[n, m]}}\[Wedge]Subscript[p, j] \[Element] P\[Wedge]k \[Element] Z\[Wedge]n \[Element] Z\[Wedge]n > 0`
NMT	`DivisorSigma [k , n] == Product [DivisorSigma [k , Subscript [p , j]] ^ Subscript [n , j] , { j , 1 , m }] /; FactorInteger [n] == { { Subscript [p , 1] , Subscript [n , 1] } , \[Ellipsis] , { Subscript [p , m] , Subscript [n , m] } } && Element [Subscript [p , j] , Primes] && Element [k , Integers] && Element [n , Integers] && n > 0`
GT.	`DivisorSigma [k , n] == Product [DivisorSigma [k , Subscript [p , j]] ^ Subscript [n , j]] , { j , 1 , m }] /; FactorInteger [n] == { { Subscript [p , 1] , Subscript [n , 1] } , \[Ellipsis] , { Subscript [p , m] , Subscript [n , m] } } && Element [Subscript [p , j] , Primes] && Element [k , Integers] && Element [n , Integers] && n > 0`

$$\wp\left(z; g_2, g_3\right) = -\frac{\sigma(z-z_0; g_2, g_3)\sigma(z+z_0; g_2, g_3)}{\sigma(z; g_2, g_3)^2 \sigma(z_0; g_2, g_3)^2} /; z_0 = \wp^{-1}\left(0; g_2, g_3\right) \qquad \text{(B.2)}$$

LATEX	`\wp \left(z ; g _ 2 , g _ 3 \right) = - \frac { \sigma \left(z - z _ 0 ; g _ 2 , g _ 3 \right) \sigma \left(z + z _ 0 ; g _ 2 , g _ 3 \right) } { \sigma \left(z ; g _ 2 , g _ 3 \right) { } ^ 2 \sigma \left(z _ 0 ; g _ 2 , g _ 3 \right) { } ^ 2 } \text{/;} z _ 0 = \wp ^ { - 1 } \left(0 ; g _ 2 , g _ 3 \right)`
Mat.	`WeierstrassP[z, {Subscript[g, 2], Subscript[g, 3]}] == -((\[Sigma][z - Subscript[z, 0]; Subscript[g, 2], Subscript[g, 3]] \[Sigma][z + Subscript[z, 0]; Subscript[g, 2], Subscript[g, 3]])/(\[Sigma][z; Subscript[g, 2], Subscript[g, 3]] \[Null]^2 \[Sigma][Subscript[z, 0]; Subscript[g, 2], Subscript[g, 3]] \[Null]^2)) /; Subscript[z, 0] == InverseWeierstrassP[0, {Subscript[g, 2], Subscript[g, 3]}]`
NMT	`WeierstrassP [z , { Subscript [g , 2] , Subscript [g , 3] }] == - (WeierstrassSigma [z - Subscript [z , 0] , { Subscript [g , 2] , Subscript [g , 3] }] WeierstrassSigma [z + Subscript [z , 0] , { Subscript [g , 2] , Subscript [g , 3] }] / (WeierstrassSigma [z , { Subscript [g , 2] , Subscript [g , 3] }] ^ 2 WeierstrassSigma [Subscript [z , 0] , { Subscript [g , 2] , Subscript [g , 3] }] ^ 2)) /; Subscript [z , 0] == InverseWeierstrassP [0 , { Subscript [g , 2] , Subscript [g , 3] }]`
GT.	`WeierstrassP [z , { Subscript [g , 2] , Subscript [g , 3] }] == - ((WeierstrassSigma [z - Subscript [z , 0] , { Subscript [g , 2] , Subscript [g , 3] }] WeierstrassSigma [z + Subscript [z , 0] , { Subscript [g , 2] , Subscript [g , 3] }]) / (WeierstrassSigma [z , { Subscript [g , 2] , Subscript [g , 3] }] ^ 2 WeierstrassSigma [Subscript [z , 0] , { Subscript [g , 2] , Subscript [g , 3] }] ^ 2)) /; Subscript [z , 0] == InverseWeierstrassP [0 , { Subscript [g , 2] , Subscript [g , 3] }]`

Table 6: **[2/3]** Equations including the tokenized LaTeX input, the (optional) interpretation by Mathematica (Mat.), our translation (NMT), and the ground truth (GT.). (NMT+GT.) denotes that our translation is equal to the ground truth.

$$Q_4^9(z) = -\frac{\texttt{<number\_17>}\big(3+\texttt{<number\_04>}z^2+\texttt{<number\_21>}z^4\big)}{(z-1)^{9/2}(z+1)^{9/2}}$$ (B.3)

LaTeX
```
Q _ 4 ^ 9 ( z ) = - \frac { <number_17> \left( 3 + <number_04> z ^ 2 +
<number_21> z ^ 4 \right) } { ( z - 1 ) ^ { 9 / 2 } ( z + 1 ) ^ { 9 / 2 }
}
```

NMT
```
GammaRegularized [ 4 , 9 , z ] == - ( ( <number_17> ( 3 + <number_04> z ^
2 + <number_21> z ^ 4 ) ) / ( ( z - 1 ) ^ ( 9 / 2 ) ( z + 1 ) ^ ( 9 / 2 )
) )
```

GT.
```
LegendreQ [ 4 , 9 , 3 , z ] == - ( ( <number_17> ( 3 + <number_04> z ^ 2
+ <number_21> z ^ 4 ) ) / ( ( z - 1 ) ^ ( 9 / 2 ) ( z + 1 ) ^ ( 9 / 2 ) )
)
```

$$\wp(z_1 \pm z_2; g_2, g_3) = \frac{(\wp(z_1;g_2,g_3)+\wp(z_2;g_2,g_3))\big(2\wp(z_1;g_2,g_3)\wp(z_2;g_2,g_3)-\frac{g_2}{2}\big)-\big(g_3\mp\wp'(z_1;g_2,g_3)\wp'(z_2;g_2,g_3)\big)}{2(\wp(z_1;g_2,g_3)-\wp(z_2;g_2,g_3))^2}$$ (B.4)

LaTeX
```
\wp \left( z _ 1 \pm z _ 2 ; g _ 2 , g _ 3 \right) = \frac { \left( \wp
\left( z _ 1 ; g _ 2 , g _ 3 \right) + \wp \left( z _ 2 ; g _ 2 , g _ 3
\right) \right) \left( 2 \wp \left( z _ 1 ; g _ 2 , g _ 3 \right) \wp
\left( z _ 2 ; g _ 2 , g _ 3 \right) - \frac { g _ 2 } { 2 } \right) -
\left( g _ 3 \mp \wp ' \left( z _ 1 ; g _ 2 , g _ 3 \right) \wp ' \left(
z _ 2 ; g _ 2 , g _ 3 \right) \right) } { 2 \left( \wp \left( z _ 1 ; g
_ 2 , g _ 3 \right) - \wp \left( z _ 2 ; g _ 2 , g _ 3 \right) \right) {
} ^ 2 }
```

NMT
```
WeierstrassP [ Subscript [ z , 1 ] \[PlusMinus] Subscript [ z , 2 ] ,
{ Subscript [ g , 2 ] , Subscript [ g , 3 ] } ] == ( ( WeierstrassP [
Subscript [ z , 1 ] , { Subscript [ g , 2 ] , Subscript [ g , 3 ] } ] +
WeierstrassP [ Subscript [ z , 2 ] , { Subscript [ g , 2 ] , Subscript [
g , 3 ] } ] ) / ( 2 ( WeierstrassP [ Subscript [ z , 1 ] , { Subscript
[ g , 2 ] , Subscript [ g , 3 ] } ] WeierstrassP [ Subscript [ z , 2 ]
, { Subscript [ g , 2 ] , Subscript [ g , 3 ] } ] - Subscript [ g , 2
] / 2 ) - ( Subscript [ g , 3 ] , WeierstrassPPrime [ Subscript [ z , 1
] , { Subscript [ g , 2 ] , Subscript [ g , 3 ] } ] WeierstrassPPrime
[ Subscript [ z , 2 ] , { Subscript [ g , 2 ] , Subscript [ g , 3 ] }
] ) ) / ( 2 ( WeierstrassP [ Subscript [ z , 1 ] , { Subscript [ g , 2
] , Subscript [ g , 3 ] } ] - WeierstrassP [ Subscript [ z , 2 ] , {
Subscript [ g , 2 ] , Subscript [ g , 3 ] } ] ) )
```

GT.
```
WeierstrassP [ Subscript [ z , 1 ] \[PlusMinus] Subscript [ z , 2 ] , {
Subscript [ g , 2 ] , Subscript [ g , 3 ] } ] == ( 1 / ( 2 ( WeierstrassP
[ Subscript [ z , 1 ] , { Subscript [ g , 2 ] , Subscript [ g , 3 ]
} ] - WeierstrassP [ Subscript [ z , 2 ] , { Subscript [ g , 2 ] ,
Subscript [ g , 3 ] } ] ) ^ 2 ) ) ( ( WeierstrassP [ Subscript [ z ,
1 ] , { Subscript [ g , 2 ] , Subscript [ g , 3 ] } ] + WeierstrassP
[ Subscript [ z , 2 ] , { Subscript [ g , 2 ] , Subscript [ g , 3 ] }
] ) ( 2 WeierstrassP [ Subscript [ z , 1 ] , { Subscript [ g , 2 ] ,
Subscript [ g , 3 ] } ] WeierstrassP [ Subscript [ z , 2 ] , { Subscript
[ g , 2 ] , Subscript [ g , 3 ] } ] - ( 1 / 2 ) Subscript [ g , 2 ] ) -
( Subscript [ g , 3 ] \[MinusPlus] WeierstrassPPrime [ Subscript [ z , 1
] , { Subscript [ g , 2 ] , Subscript [ g , 3 ] } ] WeierstrassPPrime [
Subscript [ z , 2 ] , { Subscript [ g , 2 ] , Subscript [ g , 3 ] } ] ) )
```

Table 7: **[3/3]** Equations including the tokenized LaTeX input, the (optional) interpretation by Mathematica (Mat.), our translation (NMT), and the ground truth (GT.). (NMT+GT.) denotes that our translation is equal to the ground truth.

$$\delta(0, 0, 0, 0) = 1 \tag{B.5}$$

LaTeX	`\delta ( 0 , 0 , 0 , 0 ) = 1`
Mat.	`\[Delta][0, 0, 0, 0] == 1`
NMT+GT.	`DiscreteDelta [ 0 , 0 , 0 , 0 ] == 1`

$$\sigma_k(2) = 1 + 2^k \tag{B.6}$$

LaTeX	`\sigma _ k ( 2 ) = 1 + 2 ^ k`
Mat.	`Subscript[\[Sigma], k][2] == 1 + 2^k`
NMT+GT.	`DivisorSigma [ k , 2 ] == 1 + 2 ^ k`

$$\int \sin(az) J_\nu(az) dz = \frac{z(az)_3^{1+\nu} F_4\left(\frac{3}{4}+\frac{\nu}{2},1+\frac{\nu}{2},\frac{5}{4}+\frac{\nu}{2};\frac{3}{2},2+\frac{\nu}{2},1+\nu,\frac{3}{2}+\nu;-a^2 z^2\right)}{2^\nu((2+\nu)\Gamma(1+\nu))} \tag{B.7}$$

LaTeX	**`\int \sin ( a z ) J _ { \nu } ( a z ) d z`** ` = \frac { z (a z) ^ { 1 + \nu } _ 3 F _ 4 \left(\frac { 3 } { 4 } + \frac { \nu } { 2 } , 1 + \frac { \nu } { 2 } , \frac { 5 } { 4 } + \frac { \nu } { 2 } ; \frac { 3 } { 2 } , 2 + \frac { \nu } { 2 } , 1 + \nu , \frac { 3 } { 2 } + \nu ; - a ^ 2 z ^ 2 \right) } { 2 ^ { \nu } ((2 + \nu) \Gamma (1 + \nu)) }`
Mat.	Error; left half cannot be interpreted. Interpretation of right half: `(z \!\(\*SubsuperscriptBox[\(az\), \(3\), \(1 + \[Nu]\)]\) Subscript[F, 4][3/4 + \[Nu]/2, 1 + \[Nu]/2, 5/4 + \[Nu]/2; 3/2, 2 + \[Nu]/2, 1 + \[Nu], 3/2 + \[Nu]; -a^2 z^2])/(2^\[Nu] ((2 + \[Nu]) Gamma[1 + \[Nu]]))`
NMT+GT.	**`Integrate [ Sin [ a z ] BesselJ [ \[Nu] , a z ] , z ]`** ` == ( z ( a z ) ^ ( 1 + \[Nu] ) HypergeometricPFQ [ { 3 / 4 + \[Nu] / 2 , 1 + \[Nu] / 2 , 5 / 4 + \[Nu] / 2 } , { 3 / 2 , 2 + \[Nu] / 2 , 1 + \[Nu] , 3 / 2 + \[Nu] } , ( - a ^ 2 ) z ^ 2 ] ) / 2 ^ \[Nu] / ( ( 2 + \[Nu] ) Gamma [ 1 + \[Nu] ] ) )`

$$\wp(z; g_2, g_3) = \frac{1}{z^2} + \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \text{If}\left[\{m, n\} = \{0, 0\}, 0, \frac{1}{(z-2m\omega_1-2n\omega_3)^2} - \frac{1}{(2m\omega_1+2n\omega_3)^2}\right] \tag{B.8}$$

LaTeX	`\wp \left(z ; g _ 2 , g _ 3 \right) = \frac { 1 } { z ^ 2 } + \sum _ { m = - \infty } ^ { \infty } \sum _ { n = - \infty } ^ { \infty } \text{If} \left[\{ m , n \} = \{ 0 , 0 \} , 0 , \frac { 1 } { \left(z - 2 m \omega _ 1 - 2 n \omega _ 3 \right) { } ^ 2 } - \frac { 1 } { \left(2 m \omega _ 1 + 2 n \omega _ 3 \right) { } ^ 2 } \right]`
NMT+GT.	`WeierstrassP [z , { Subscript [g , 2] , Subscript [g , 3] }] == 1 / z ^ 2 + Sum [If [{ m , n } == { 0 , 0 } , 0 , 1 / (z - 2 m Subscript [\[Omega] , 1] - 2 n Subscript [\[Omega] , 3]) ^ 2 - 1 / (2 m Subscript [\[Omega] , 1] + 2 n Subscript [\[Omega] , 3]) ^ 2] , { m , - Infinity , Infinity } , { n , - Infinity , Infinity }]`

C   Network Ablation Studies

Ablation studies based on the LATEX→Mathematica translation model. The concrete results for the analysis are displayed in Tables 8–11. For the tables, let C$sx$$n$ denote a convolutional encoder and equal decoder with state size s, kernel size 3, and n consecutive layers. Let C$s$ks$kx$$n$ be defined according to the previous definition but with a kernel size of k. Further, let y-z be the concatenation of three elements: y, a fully connected affine layer translating between the state sizes of y and z, and z. Let the embedding size equal the state size of the first layer. For accuracy, we used the exact match accuracy on the validation set of the LATEX→Mathematica translation.

Table 8: Experiments on mixed and constant state/embedding sizes.

Model	Acc.
C256x8	86.4%
C256x12	88.3%
C512x6-C768x4-C1024x3- C2048x1-C4096x1	88.6%
C512x4-C1024x4	91.2%
C512x6-C768x4-C1024x2	91.6%
C512x8	91.9%
C512x4-C1024x8	92.3%
C512x8-C1024x4	92.7%
C512x20	93.0%
C512x12	94.9%

Table 9: Additional experiments (based on C512x8).

Modification	Acc.
Substitute Numbers	95.0%
Single-digit tokens	92.7%
Training bias towards short formulae	94.8%
Input dict. $\neq$ output dict.	95.0%

Table 10: Experiments on different numbers of layers.

Model	C512x8	C512x9	C512x10	C512x11	C512x12	C512x13
Acc.	94.3%	94.5%	94.7%	95.1%	95.0%	94.8%

Table 11: Experiments comparing kernel sizes (including number of parameters).

Model	C512ks3x8	C512ks5x8	C512ks7x8	C512ks5x10	C512ks3x11	C512ks5x11
Acc.	94.3%	95.2%	94.1%	94.4%	95.1%	95.1%
Num. of param.	32 671 200	49 448 416	66 225 632	60 995 040	43 699 680	66 768 352