Multi-Operational Mathematical Derivations in Latent Space

Marco Valentino[1], **Jordan Meadows**[2], **Lan Zhang**[2], **André Freitas**[1,2,3]
[1]Idiap Research Institute, Switzerland
[2] Department of Computer Science, University of Manchester, United Kingdom
[3] Cancer Biomarker Centre, CRUK Manchester Institute, United Kingdom
{marco.valentino, andre.freitas}@idiap.ch
{jordan.meadows, lan.zhang-6}@manchester.ac.uk

Abstract

This paper investigates the possibility of approximating multiple mathematical operations in latent space for expression derivation. To this end, we introduce different multi-operational representation paradigms, modelling mathematical operations as explicit geometric transformations. By leveraging a symbolic engine, we construct a large-scale dataset comprising 1.7M derivation steps stemming from 61K premises and 6 operators, analysing the properties of each paradigm when instantiated with state-of-the-art neural encoders. Specifically, we investigate how different encoding mechanisms can approximate expression manipulation in latent space, exploring the trade-off between learning different operators and specialising within single operations, as well as the ability to support multi-step derivations and out-of-distribution generalisation. Our empirical analysis reveals that the multi-operational paradigm is crucial for disentangling different operators, while discriminating the conclusions for a single operation is achievable in the original expression encoder. Moreover, we show that architectural choices can heavily affect the training dynamics, structural organisation, and generalisation of the latent space, resulting in significant variations across paradigms and classes of encoders[1].

1 Introduction

To what extent are neural networks capable of mathematical reasoning? This question has led many researchers to propose various methods to train and test neural models on different math-related tasks, such as math word problems, theorem proving, and premise selection (Lu et al., 2023; Meadows and Freitas, 2023; Mishra et al., 2022a; Ferreira et al., 2022; Ferreira and Freitas, 2020; Welleck et al., 2021; Valentino et al., 2022; Mishra et al., 2022b; Petersen et al., 2023). These methods aim to investigate how neural architectures learn and generalise mathematical concepts and symbolic rules, and how they cope with characteristic challenges of mathematical inference, such as abstraction, compositionality, and systematicity (Welleck et al., 2022; Mishra et al., 2022a).

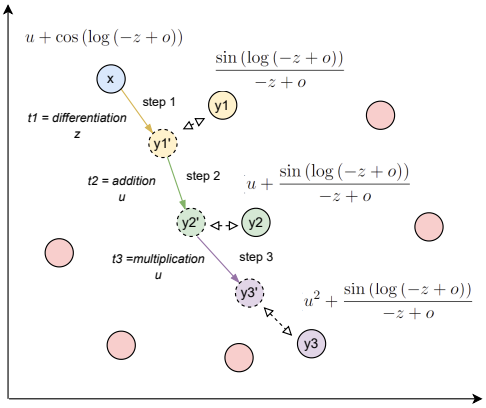


Figure 1: *Can neural encoders learn to approximate multiple mathematical operators in latent space?* Given a premise x, we investigate the problem of applying a sequence of latent operations $(t_1, \ldots, t_n)$ to derive valid mathematical expressions $(y_1, \ldots, y_n)$.

In general, a key challenge in neural mathematical reasoning is to represent expressions and formulae into a latent space to enable the application of multiple operations in specific orders under contextual constraints. Existing methods, however, typically focus on single-operational inference – i.e., optimising a latent space to approximate a specific mathematical operation (Lee et al., 2019; Lample and Charton, 2019; Welleck et al., 2022). Encoding multiple operations in the same latent space, therefore, remains an unexplored challenge that will likely require the development of novel mechanisms and representational paradigms.

To investigate this problem, this paper focuses

[1]Code & data available at: https://github.com/neuro-symbolic-ai/latent_mathematical_reasoning

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 1446–1458
June 16-21, 2024 ©2024 Association for Computational Linguistics

on *equational reasoning*, intended as *the derivation of expressions from premises via the sequential application of specialised mathematical operations* (i.e., addition, subtraction, multiplication, division, integration, differentiation). As derivations represent the workhorse of applied mathematical reasoning (including derivations in physics and engineering), projecting expressions and operators into a well-organised geometric space can unveil a myriad of applications, unlocking the approximation of mathematical solutions that are multiple steps apart within the embedding space via distance metrics and vector operations.

Specifically, this paper posits the following overarching research questions: *RQ1:"How can different representational paradigms and encoding mechanisms support expression derivation in latent space?"; RQ2:"What is the representational trade-off between generalising across different mathematical operations and specialising within single operations?"; RQ3:"To what extent can different encoding mechanisms enable multi-step derivations through the sequential application and functional composition of latent operators?"; RQ4:"To what extent can different encoding mechanisms support out-of-distribution generalisation?"*

To answer these questions, we investigate joint-embedding predictive architectures (LeCun, 2022) by introducing different multi-operational representation paradigms (i.e., *projection* and *translation*) to model mathematical operations as explicit geometric transformations within the latent space. Moreover, by leveraging a symbolic engine (Meurer et al., 2017), we build a large-scale dataset containing 1.7M derivation steps which span diverse mathematical expressions and operations. To understand the impact of different encoding schemes on equational reasoning, we instantiate the proposed architectures with state-of-the-art neural encoders, including Graph Neural Networks (GNNs) (Hamilton et al., 2017; Kipf and Welling, 2016), Convolutional Neural Networks (CNNs) (Li et al., 2021; Kim, 2014), Recurrent Neural Networks (RNNs) (Yu et al., 2019; Hochreiter and Schmidhuber, 1996), and Transformers (Vaswani et al., 2017), analysing the properties of the latent spaces and the ability to support multi-step derivations and generalisation.

Our empirical evaluation reveals that the multi-operational paradigm is crucial for disentangling different mathematical operators (i.e., *cross-operational inference*), while the discrimination of the conclusions for a single operation (i.e., *intra-operational inference*) is achievable in the original expression encoder. Moreover, we show that architectural choices can heavily affect the training dynamics and the structural organisation of the latent space, resulting in significant variations across paradigms and classes of encoders.

Overall, we conclude that the translation paradigm can result in a more fine-grained and smoother optimisation of the latent space, which better supports cross-operational inference and enables a more balanced integration. Regarding the encoders, we found that sequential models achieve more robust performance when tested on multi-step derivations, while graph-based encoders, on the contrary, exhibit better generalisation to out-of-distribution examples.

2 Multi-Operational Derivations

Given a premise x – i.e., a mathematical expression including variables and constants, and a set of operations $T = \{t_1, t_2, \dots t_n\}$ – e.g., addition, multiplication, differentiation, etc., we investigate the extent to which a neural encoder can approximate a mathematical function $f(x, t_i; V) = Y_{t_i}$ that takes the premise x and any operation $t_i \in T$ as inputs, and produces the set of valid expressions $Y_{t_i} = \{y_1, y_2, \dots, y_m\}$ derivable from x via t_i given $v \in V$, where V is a predefined set of operands such that $y_j = t_i(x, v_j)$. In this work, we focus on *atomic operations* in which V includes symbols representing variables.

For example, consider the following premise x:

$$u + \cos\left(\log\left(-z + o\right)\right)$$

If the set of operands is $V = \{z, u\}$, and the operation t_i is *addition*, then the application of t_i to x should result in the set of expressions:

$$Y_{add} = \{z + u + \cos\left(\log\left(-z + o\right)\right), \dots,$$
$$2u + \cos\left(\log\left(-z + o\right)\right)\}$$

Instead, if t_i is *differentiation*, then $f(x, t_i; V)$ should result in a different set:

$$Y_{diff} = \{\frac{\sin\left(\log\left(-z + o\right)\right)}{-z + o}, \dots, 1\}$$

Notably, the recursive application of f to any of the expressions in Y_{t_i} can generate a new set of conclusions derivable from x in multiple steps.

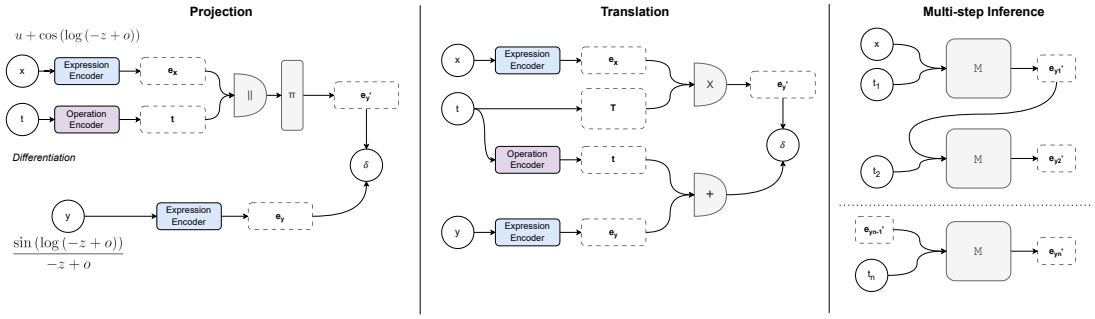


Figure 2: Overview of the proposed joint-embedding predictive architectures for latent multi-operational derivation (left). Schematic workflow for multi-step inference and latent propagation of mathematical operations (right).

Here, the constraint we are interested in, is that Y_{add} and Y_{diff} should be derived via a *single expression encoder* that maps expressions into a vector space and, at the same time, enables a multi-step propagation of latent operations.

2.1 Architectures

To model latent mathematical operations, we investigate the use of joint-embedding predictive architectures (LeCun, 2022). In particular, we introduce two multi-operational paradigms based on *projection* and *translation* to learn the representation of expressions and mathematical operators and model an atomic derivation step as an explicit *geometric transformation*. Figure 2 shows a schematic representation of the architectures.

In general, both projection and translation employ an *expression encoder* to map the premise x and a plausible conclusion y into vectors, along with an *operation encoder* that acts as a *latent prompt* $\mathbf{t}$ to discriminate between operators. The goal is then to predict the embedding of a valid conclusion $\mathbf{e}_y$ by applying a transformation to the premise embedding $\mathbf{e}_x$ conditioned on $\mathbf{t}$. Therefore, the two paradigms mainly differ in how expression and operation embeddings are combined to approximate the target results. This setup enables *multi-step inference* since the predicted embedding $\mathbf{e}'_y$ can be recursively interpreted as a premise representation for the next iteration (Figure 2, right).

Projection. The most intuitive solution to model latent mathematical operations is to employ a *projection layer* (Lee et al., 2019). In this case, the premise x and the operator t are first embedded using the respective encoders, which are then fed to a dense predictive layer π to approximate the target conclusion $\mathbf{e}_y$. The overall objective function can then be formalised as follows:

$$\phi(x, t, y) = -\delta(\pi(\mathbf{t} \| \mathbf{e}_x), \mathbf{e}_y)^2 \quad (1)$$

Where δ is a distance function, and π represents the dense projection applied to the concatenation $\|$ of $\mathbf{t}$ and $\mathbf{e}_x$. While many options are available, we implement π using a linear layer to better investigate the representation power of the underlying expression encoder.

Translation. Inspired by research on multi-relational graph embeddings (Bordes et al., 2013; Balazevic et al., 2019; Valentino et al., 2023), we frame mathematical inference as a *multi-relational representation learning* problem. In particular, it is possible to draw a direct analogy between entities and relations in a knowledge graph and mathematical operations. Within the scope of the task, as defined in Section 2, the application of a general operation can be interpreted as a relational triple $< x, t, y >$, in which a premise expression x corresponds to the subject entity, a conclusion y corresponds to the object entity, and the specific operation type t represents the semantic relation between entities. Following this intuition, we formalise the learning problem via a translational objective:

$$\phi(x, t, y) = -\delta(\mathbf{T}\mathbf{e}_x, \mathbf{e}_y + \mathbf{t})^2 \quad (2)$$

Where δ is a distance function, $\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{t}$, are the embeddings of premise expression, conclusion and operation, and $\mathbf{T}$ is a diagonal operation matrix.

2.2 Data Generation

We generate synthetic data to support the exploration of the above architectures, inspired by a recent approach that relies on a symbolic engine to generate equational reasoning examples (Meadows et al., 2023b). In particular, we use SymPy (Meurer et al., 2017) to construct a dataset containing expressions in both LaTeX and SymPy surface forms.

Here, premises and variables are input to 6 operations to generate further expressions, presently focusing on differentiation, integration, addition, subtraction, multiplication, and division. Concrete examples of entries in the dataset are reported in the Appendix.

Premises. To generate a premise expression, a set of symbols is first sampled from a vocabulary. Subsequently, an initial operator is applied to symbols to generate an expression via the SymPy engine. To generate more complex expressions, this process is repeated iteratively for a fixed number of steps. This process is formalised in Algorithm 1 (see Appendix). The final dataset includes 61K premises each containing between 2 to 5 variables.

Applying Operations to Premises. For a given premise, a set of operand variables (denoted by V in Section 2) are sampled from the vocabulary and added to the set of symbols that comprise the premise. All valid combinations of premise and operands are then input to each operator (via SymPy) to generate conclusions derivable via atomic derivation steps. The resulting dataset contains a total of 1.7M of such atomic steps. This data is used to train and evaluate models on single-step inference before testing generalisation capabilities to multiple steps.

Multi-Step Derivations. To test the models' ability to derive expressions obtained after the sequential application of operations, we randomly sample 5K premises from the single-step dataset described above and iteratively apply up to 6 operations to each premise using a randomly sampled variable operand from the vocabulary for each step. We adopt this methodology to generate a total of 2.7K multi-step examples.

2.3 Expression Encoders

Thanks to their generality, the multi-operational architectures can be instantiated with different classes of expression encoders. In particular, we experiment with both *graph-based* and *sequential* models, exploring embeddings with different dimensions (i.e., 300, 512, and 768). The graph-based encoders are trained on operation trees extracted from the SymPy representation, while the sequential models are trained on LaTeX expressions. We adopted the following expression encoders in our experiments:

Graph Neural Networks (GNNs). GNNs have been adopted for mathematical inference thanks to their ability to capture explicit structural information (Lee et al., 2019). Here, we consider different classes of GNNs to experiment with models that can derive representations from operation trees. Specifically, we employ a 6-layer GraphSage[2](Hamilton et al., 2017) and Graph Convolutional Network (GCN)[3] (Kipf and Welling, 2016) to investigate transductive and non-transductive methods. To build the operation trees, we directly parse the SymPy representation described in Appendix A.

Convolutional Neural Networks (CNNs). CNNs represent an effective class of models for mathematical representation learning thanks to their translation invariance property that can help localise recurring symbolic patterns within expressions (Petersen et al., 2023). Here, we employ a 1D CNN architecture typically used for text classification tasks (Kim, 2014), with three filter sizes 3, 4, and 5, each with 100 filters.

Recurrent Neural Networks (RNNs). Due to the sequential nature of mathematical expressions, we experiment with RNNs that have been successful in modelling long-range dependencies for sentence representation (Yu et al., 2019; Hochreiter and Schmidhuber, 1996). In particular, we employ a Long-Short Term Memory (LSTM) network with 2 layers.

Transformers. Finally, we experiment with a Transformer encoder with 6 and 8 attention heads and 6 layers, using a configuration similar to the one proposed by Vaswani et al. (2017)[4]. Differently from other models, Transformers use the attention mechanism to capture implicit relations between tokens, allowing, at the same time, experiments with a larger number of trainable parameters.

2.4 Operation Encoders

The operation encoders are implemented using a lookup table similar to word embeddings (Mikolov et al., 2013), where each entry corresponds to the vector of a mathematical operator. We experiment

[2] https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.models.GraphSAGE.html

[3] https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.models.GCN.html

[4] https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html

	MAP	Hit@1	Hit@3	MAP	Hit@1	Hit@3	Avg. MAP
Projection (One-hot)		**Cross-op.**			**Intra-op.**		
GCN	74.07	78.35	88.88	93.34	97.81	99.01	83.70
GraphSAGE	**83.89**	**88.43**	**96.70**	93.00	97.45	98.71	**88.44**
CNN	69.61	76.98	95.20	92.43	97.18	98.63	81.02
LSTM	71.40	73.50	90.08	**93.21**	**98.01**	**99.35**	82.30
Transformer	49.35	46.30	63.00	91.66	96.65	98.38	70.50
Projection (Dense)							
GCN	78.25	82.50	92.81	93.43	97.91	99.08	85.84
GraphSAGE	81.05	83.91	94.38	93.18	97.81	98.93	87.11
CNN	**82.57**	**91.40**	**98.50**	92.62	97.15	99.18	**87.59**
LSTM	77.17	81.96	93.73	**93.68**	**98.48**	**99.36**	85.42
Transformer	71.51	77.08	89.43	92.23	97.30	98.53	81.87
Translation							
GCN	85.89	94.73	98.85	90.10	92.45	95.61	87.99
GraphSAGE	88.15	96.31	99.25	90.68	94.51	96.88	89.41
CNN	84.72	94.66	98.70	90.17	93.98	97.96	87.44
LSTM	**89.85**	**96.70**	**99.35**	89.74	94.60	97.91	**89.79**
Transformer	86.64	95.78	98.83	**90.93**	**96.05**	**99.73**	88.78

Table 1: Overall performance of different neural encoders and methods for encoding multiple mathematical operations (i.e., integration, differentiation, addition, difference, multiplication, division) in the latent space.

with *dense*[5] embeddings for the translation model and instantiate the projection architecture with both *dense* and *one-hot*[6] embeddings. The translation model requires the operation embeddings to be the same size as the expression embeddings, admitting, therefore, only dense representations.

3 Training Details

As the models are trained to predict a target embedding, the main goal during optimisation is to avoid a *representational collapse* in the expression encoder. To this end, we opted for a *Multiple Negatives Ranking (MNR)* loss with in-batch negative examples (Henderson et al., 2017). This technique allows us to sidestep the explicit selection of the negative sample, enabling a smoother optimisation of the latent space. We trained the models on a total of 12.800 premise expressions with 24 positive examples each derived from the application of 6 operations (see Section 2.2). This produces over 307.200 training instances composed of premise x, operation t, and conclusion y. The models are then trained for 32 epochs with a batch size of 64 (with in-batch random negatives). We found that the best results are obtained with a learning rate of 1e-5.

[5] https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html

[6] https://pytorch.org/docs/stable/generated/torch.nn.functional.one_hot.html

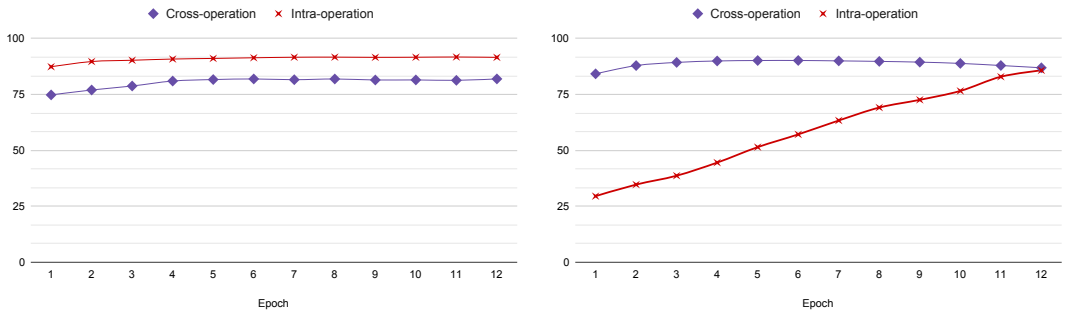
4 Empirical Evaluation

4.1 Empirical Setup

We evaluate the performance of different representational paradigms and expression encoders by building held-out dev and test sets. In particular, to assess the structural organisation of the latent space, we frame the task of multi-operational inference as an expression retrieval problem. Given a premise x, an operation t, a sample of positive conclusions $P = \{p_1, \ldots, p_n\}$, and a sample of negative conclusions $N = \{n_1, \ldots, n_m\}$, we adopt the models to predict an embedding $\mathbf{e}'_y$ (Section 2.1) and employ a distance function δ to rank all the conclusions in $P \cup N$ according to their similarity with $\mathbf{e}'_y$. We implement δ using cosine similarity, and construct two evaluation sets to assess complementary inferential properties, namely:

Cross-operational Inference. A model able to perform multi-operational inference should discriminate between the results of different operations applied to the same premise. Therefore, given a premise x and an operation t (e.g., addition), we construct the negative set N by selecting the positive conclusions resulting from the application of different operations (e.g., differentiation, subtraction) to the same premise x. This set includes a total of 4 positive and 20 negative examples (extracted from the remaining 5 operations) for each premise-operation pair (for a total of 3k dev and 6k test instances).

Intra-operational Inference. While we want the models to discriminate between different operators,



(a) Projection

(b) Translation

Figure 3: Typical training dynamics of different multi-operational paradigms (MAP on the dev set).

a well-optimised latent space should still preserve the ability to predict the results of a single operation applied to different premises. Therefore, given a premise x and an operation t, we construct the negative set N by selecting the positive conclusions resulting from the application of the same operation t to a different sample of premises. This set includes a total of 4 positive and 20 negative examples (extracted from 5 random premises) for each premise-operation pair (for a total of 3k dev and 6k test instances).

Metrics. The models are evaluated using Mean Average Precision (MAP) and Hit@k. Hit@k measures the percentage of test instances in which at least one positive conclusion is ranked within the top k positions. MAP, on the other hand, measures the overall ranking. We use the average MAP between cross-operational and intra-operational sets (dev) as a criterion for model selection.

4.2 Results

Table 1 shows the performance of different encoders and paradigms on the test sets (i.e., evaluating the best models from the dev set, see Table 3). We can derive the following conclusions:

The translation mechanism improves cross-operational inference. The models that use the translation method consistently outperform the models that use the projection method on the cross-operational inference task. This indicates that the translation paradigm can better capture the semantic relations between different operations and preserve them in the latent space. This is attested by the significant improvement achieved by different encoders, involving both graph-based and sequential architectures (e.g., +15.13% and +7.64% for Transformers and GCN respectively).

Trade-off between cross-operational and intra-operational inference. The models that excel at cross-operational inference tend to achieve lower performance on the intra-operational set. This suggests that there is a tension between generalising across different operations and specialising within each operation. Moreover, the results suggest that intra-operational inference represents an easier problem for neural encoders that can be achieved already with sparse multi-operational methods (i.e., models using one-hot projection can achieve a MAP score above 90%).

LSTMs and GraphSAGE achieve the best performance. LSTMs achieve the highest average MAP score, followed by GraphSAGE. These results demonstrate that LSTMs and GraphSAGE can balance between generalisation and specialisation, and leverage both sequential and graph-based information to encode mathematical operations. Moreover, we observe that graph-based models and CNNs tend to exhibit more stable performance across different representational paradigms (e.g., GraphSage achieve an average improvement of 2.3%), while LSTMs and Transformers achieve balanced results only with the translation mechanism (i.e., with an average improvement of 4.37% and 6.91% respectively).

Model size alone does not explain inference performances. The Transformer model, which has the largest number of parameters, exhibits a lower average MAP score (with the projection mechanism in particular). This implies that simply increasing the model complexity or capacity does not guarantee better results (see Table 3 for additional details) and may compromise operational control in the latent space. This suggests that model architec-

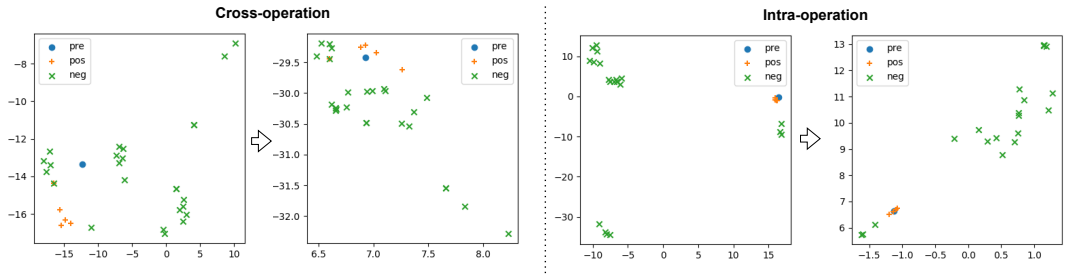


Figure 4: 2D projection of the latent space before and after an operation-specific transformation. The visualization supports the crucial role of the multi-operational paradigm for *cross-operational* inference, showing, at the same time, that *intra-operational* inference concerns larger regions and can be achieved in the original expression encoder.

	Cross		Intra	
Proj. (1-hot)	$\delta(\mathbf{e}_x, \mathbf{e}_y)$	$\delta(\mathbf{e}'_y, \mathbf{e}_y)$	$\delta(\mathbf{e}_x, \mathbf{e}_y)$	$\delta(\mathbf{e}'_y, \mathbf{e}_y)$
GCN	00.00	20.87	84.46	84.82
GraphSAGE	00.00	25.71	84.00	85.77
CNN	00.00	19.64	85.81	85.86
LSTM	00.00	24.11	86.12	84.87
Transformer	00.00	16.52	**88.20**	**86.80**
Proj. (Dense)	$\delta(\mathbf{e}_x, \mathbf{e}_y)$	$\delta(\mathbf{e}'_y, \mathbf{e}_y)$	$\delta(\mathbf{e}_x, \mathbf{e}_y)$	$\delta(\mathbf{e}'_y, \mathbf{e}_y)$
GCN	00.00	22.88	84.38	85.25
GraphSAGE	00.00	25.47	84.96	86.01
CNN	00.00	23.13	83.38	82.84
LSTM	00.00	25.66	84.80	83.44
Transformer	00.00	21.54	86.22	83.80
Translation	$\delta(\mathbf{e}_x, \mathbf{e}_y)$	$\delta(\mathbf{e}'_y, \mathbf{e}_y + \mathbf{t})$	$\delta(\mathbf{e}_x, \mathbf{e}_y)$	$\delta(\mathbf{e}'_y, \mathbf{e}_y + \mathbf{t})$
GCN	00.00	11.85	-07.13	39.76
GraphSAGE	00.00	11.37	-01.45	40.98
CNN	00.00	05.23	12.32	33.68
LSTM	00.00	40.20	-00.46	51.46
Transformer	00.00	**43.38**	03.07	69.14

Table 2: Latent separation of positive and negative examples before (i.e., $\delta(\mathbf{e}_x, \mathbf{e}_y)$) and after (i.e., $\delta(\mathbf{e}'_y, \mathbf{e}_y)$) applying an *operation-specific transformation*.

ture and the encoding method are more important factors for learning effective representations supporting multiple mathematical operations.

4.3 Training Dynamics

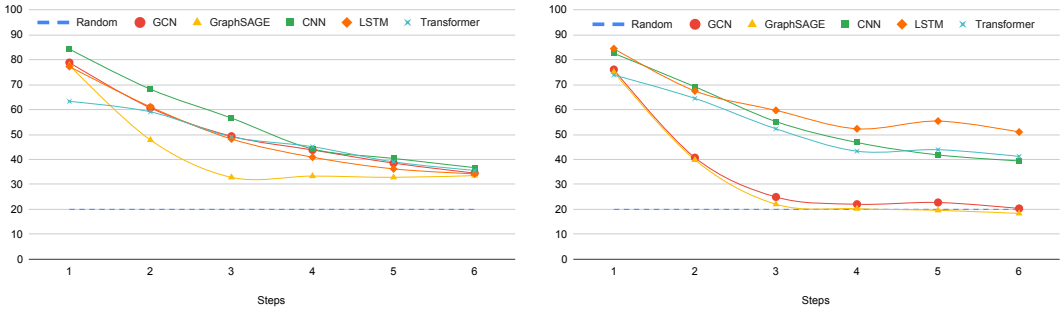
We conduct an additional analysis to investigate the training dynamics of different architectures. The graphs in Figure 3 show the typical trend for the MAP achieved at different epochs on different evaluation sets. Interestingly, we found that **the projection and translation mechanisms optimise the latent space in a different way**. The projection paradigm, in fact, prioritises performance on intra-operational inference, with a constant gap between the two sets. Conversely, the translation paradigm supports a rapid optimisation of cross-operational inference, followed by a more gradual improvement on the intra-operational set.

This behaviour can help explain the difference in performances between the models. Specifically, since cross-operational inference is about disentangling operations applied to the same premise, we hypothesise it to require a more fine-grained optimisation in localised regions of the latent space. This optimisation can be compromised when priority is given to the discrimination of different premises, which, as in the case of intra-operational inference, involves a more coarse-grained optimisation in larger regions of the space.

4.4 Latent Space Analysis

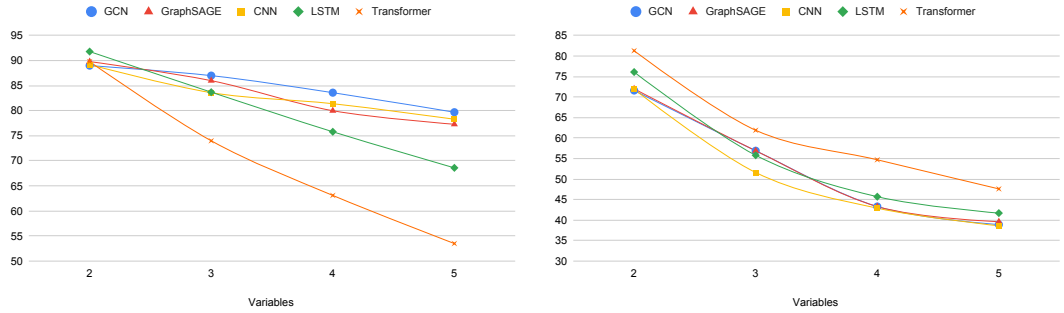
We further investigate this behaviour by measuring and visualising the latent space in the original expression encoder (i.e., computing $\delta(\mathbf{e}_x, \mathbf{e}_y)$) and after applying a transformation via the operation encoder (i.e., computing $\delta(\mathbf{e}'_y, \mathbf{e}_y)$). In particular, Table 2 reports the average difference between the cosine similarity of the premises with positive and negative examples, a measure to estimate the latent space separation, and therefore, assess how dense the resulting vector space is.

From the results, we can derive the following main observations: **(1) The separation tends to be significantly lower in the cross-operational set**, confirming that the latent space requires a more fine-grained optimisation in localised regions (Fig. 4); **(2) Cross-operational inference is not achievable without operation-specific transformations**, as confirmed by the impossibility to discriminate between positive and negative examples in the original expression encoders (i.e., $\delta(\mathbf{e}_x, \mathbf{e}_y)$, Table 2); **(3) The projection mechanism achieves intra-operational separation in the original expression encoders.** This is not true for the translation mechanism in which the transformation induced by the operation encoder is fundamental for the separation to appear; **(4) The latent space resulting from the translation model is more dense**, with values for the separation that are generally lower



(a) Projection (b) Translation

Figure 5: Multi-step derivations in latent space with different multi-operational paradigms and neural encoders.



(a) Cross-operational (b) Intra-operational

Figure 6: Length generalisation experiments by training different encoders (i.e., with translation) on premises with 2 variables, and testing on longer premises (MAP score).

when compared to the projection mechanism.

These results, combined with the performance in Table 1, confirm that the translation paradigm can result in a more fine-grained and smoother optimisation which supports performance on cross-operational inference and a more balanced integration between expression and operation encoders.

4.5 Multi-Step Inference

We investigate the behaviour of different encoders and representational paradigms when propagating latent operations for multiple steps. To experiment, we employ the architectures recursively by interpreting the predicted target embedding $\mathbf{e}'_y$ as a premise representation for the next step (see Fig. 2). In this case, we evaluate the performance using Hit@1, selecting 1 positive example and 4 negative examples for each premise and derivation step (2 for cross-operational and 2 for intra-operational).

Figure 5 shows the obtained results. We found that **the majority of the models exhibit a latent organisation that allows for a non-random propagation of latent mathematical operations**. Most of the encoders, in fact, achieve performances that are significantly above random performance after 6 latent derivation steps (with a peak of 30% improvement for LSTM + translation). Moreover, while all the models tend to decrease in performance with an increasing number of inference steps, **we observe significant differences between paradigms and classes of encoders**. Most notably, we found that **the performance of graph-based encoders tends to decrease faster, while the sequential models can obtain more stable results, in particular with the translation paradigm**. The best translation model (i.e., LSTM) achieves a Hit@1 score at 6 steps of up to 50%, that is ≈15% above the best projection architecture (i.e., CNN).

4.6 Length Generalisation

Finally, we perform experiments to test the ability of expression encoders to generalise to out-of-distribution examples. In particular, we focus on length generalisation which constitutes a notoriously hard problem for neural networks (Shen et al., 2021; Hupkes et al., 2020; Geirhos et al., 2020). To

this end, we train the models on the subset of the training set containing premises with 2 variables and assess performance on longer premises (i.e., grouping the test set according to the number of variables). Figure 6 shows the results for different encoders using the translation mechanism.

Overall, the results show **a decrease in performance as expected, demonstrating, at the same time, a notable difference between encoders on cross-operational inference**. In particular, the results suggest that **graph-based models can generalise significantly better on longer premises**, probably due to their ability to capture explicit hierarchical dependencies within the expressions. **Among the sequential models, CNNs achieve better generalisation performance**. We attribute these results to the convolution operation in CNNs which may help capture structural invariances within the expressions and allow a generalisation that is similar to GCNs.

4.7 Discussion

From the empirical evaluation, we can derive a set of takeaways for both the joint-embedding architectures and the specific expression encoders.

Regarding the architectures, our analysis suggests that the translational paradigm can result in a more fine-grained and smoother optimisation of the latent space (Figure 3 and Table 2). This has the effect of improving multi-operational inference enabling a more balanced integration of different expression encoders, with an overall better trade-off between cross-operational and intra-operational inference (Table 1). Moreover, we found that the translational paradigm can support better generalisation on multi-step inference when instantiated with sequential encoders such as Transformers, CNNs, and LSTMs (Figure 5), even when the encoders are only trained on single-step derivations.

Regarding the specific encoders, we conclude that different models have different characteristics that should inform practitioners and future research in the field. Sequential models (i.e., Transformers, CNNs, and LSTMs), possess a better ability to organise the latent space for enabling latent multi-step derivations (Figure 5). Conversely, graph-based models are more efficient (i.e., they achieve better performance using smaller operation encoders, see one-hot in Table 1) and tend to generalise better to longer expressions when trained to simpler ones (see Figure 6).

5 Related Work

The quest to understand whether neural architectures can perform mathematical reasoning has led researchers to investigate several tasks and evaluation methods (Lu et al., 2023; Meadows and Freitas, 2023; Mishra et al., 2022a; Ferreira et al., 2022; Ferreira and Freitas, 2020; Welleck et al., 2021; Valentino et al., 2022; Mishra et al., 2022b; Petersen et al., 2023). In this work, we focused on equational reasoning, a particular instance of mathematical reasoning involving the manipulation of expressions through the systematic application of specialised operations (Welleck et al., 2022; Lample and Charton, 2019; Saxton et al., 2018). In particular, our work is inspired by previous attempts to approximate mathematical reasoning entirely in latent space (Lee et al., 2019). Differently from Lee et al. (2019), we investigate the joint approximation of multiple mathematical operations for expression derivation (Lee et al. (2019) explore exclusively the rewriting operation for theorem proving). Moreover, while Lee et al. (2019) focus on the evaluation of Graph Neural Networks (Paliwal et al., 2020)), we analyse the behaviour of a diverse set of representational paradigms and neural encoders. Our data generation methodology is inspired by recent work leveraging symbolic engines and algorithms to build systematic benchmarks for neural models (Meadows et al., 2023b,a; Chen et al., 2022; Saparov et al., 2023). However, to the best of our knowledge, we are the first to construct and release a synthetic dataset to investigate multi-step and multi-operational derivations in latent space.

6 Conclusion

This paper focused on equational reasoning for expression derivation to investigate the possibility of approximating and composing multiple mathematical operations in a single latent space. Specifically, we investigated different representational paradigms and encoding mechanisms, analysing the trade-off between encoding different mathematical operators and specialising within single operations, as well as the ability to support multi-step derivations and out-of-distribution generalisation. Moreover, we constructed and released a large-scale dataset comprising 1.7M derivation steps stemming from 61K premises and 6 operators, which we hope will encourage researchers to explore future work in the field.

7 Limitations

The systematic application of mathematical operators requires reasoning at an intentional level, that is, the execution and composition of mathematical functions defined on a potentially infinite set of elements. Neural networks, on the contrary, operate at an extensional level and, by their current nature, can only approximate such functions by learning from a finite set of examples.

Due to this characteristic, this work explored architectures that are trained on expressions composed of a predefined number and set of variables (i.e., between 2 and 5) and operators (i.e., addition, subtraction, multiplication, division, integration, differentiation), and, therefore, capable of performing approximation over a finite vocabulary of symbols. Extending the architectures with a new set of operations and out-of-vocabulary symbols, therefore, would require re-training the models from scratch. Future work could investigate this limitation by exploring, for instance, transfer learning techniques and more flexible neural architectures.

For the same reason, we restricted our investigation to the encoding of atomic operations, that is, operations in which the second operand is represented by a variable. While this limitation is circumvented by the sequential application of operators in a multi-step fashion, this work did not explore the encoding of single-step operations involving more complex operands (e.g., multiplication between two expressions composed of multiple variables each). In principle, however, the evaluation presented in this work can be extended with the new synthetic data to accommodate and study different cases and setups in the future.

Acknowledgements

This work was partially funded by the Swiss National Science Foundation (SNSF) project NeuMath (200021_204617), by the EPSRC grant EP/T026995/1 entitled "EnnCore: End-to-End Conceptual Guarding of Neural Architectures" under Security for all in an AI enabled society, by the CRUK National Biomarker Centre, and supported by the Manchester Experimental Cancer Medicine Centre.

References

Ivana Balazevic, Carl Allen, and Timothy Hospedales. 2019. Multi-relational poincaré graph embeddings. *Advances in Neural Information Processing Systems*, 32.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Deborah Ferreira and André Freitas. 2020. Premise selection in natural language mathematical texts. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7365–7374.

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.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*.

Sepp Hochreiter and Jürgen Schmidhuber. 1996. Lstm can solve hard long time lag problems. *Advances in neural information processing systems*, 9.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

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

Yann LeCun. 2022. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62.

Dennis Lee, Christian Szegedy, Markus Rabe, Sarah Loos, and Kshitij Bansal. 2019. Mathematical reasoning in latent space. In *International Conference on Learning Representations*.

Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. 2021. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2023. A survey of deep learning for mathematical reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14605–14631, Toronto, Canada. Association for Computational Linguistics.

Jordan Meadows and André Freitas. 2023. Introduction to mathematical language processing: Informal proofs, word problems, and supporting tasks. *Transactions of the Association for Computational Linguistics*, 11:1162–1184.

Jordan Meadows, Marco Valentino, and Andre Freitas. 2023a. Generating mathematical derivations with large language models. *arXiv preprint arXiv:2307.09998*.

Jordan Meadows, Marco Valentino, Damien Teney, and Andre Freitas. 2023b. A symbolic framework for systematic evaluation of mathematical reasoning with transformers. *arXiv preprint arXiv:2305.12563*.

Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. 2022a. LILA: A unified benchmark for mathematical reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5807–5832, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. 2022b. Numglue: A suite of fundamental yet challenging mathematical reasoning tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3505–3523.

Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. 2020. Graph representations for higher-order logic and theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2967–2974.

Felix Petersen, Moritz Schubotz, Andre Greiner-Petter, and Bela Gipp. 2023. Neural machine translation for mathematical formulae. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11534–11550, Toronto, Canada. Association for Computational Linguistics.

Abulhair Saparov, Richard Yuanzhe Pang, Vishakh Padmakumar, Nitish Joshi, Seyed Mehran Kazemi, Najoung Kim, and He He. 2023. Testing the general deductive reasoning capacity of large language models using ood examples. *arXiv preprint arXiv:2305.15269*.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2018. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*.

Zheyan Shen, Jiashuo Liu, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu, and Peng Cui. 2021. Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*.

Marco Valentino, Danilo S Carvalho, and André Freitas. 2023. Multi-relational hyperbolic word embeddings from natural language definitions. *arXiv preprint arXiv:2305.07303*.

Marco Valentino, Deborah Ferreira, Mokanarangan Thayaparan, André Freitas, and Dmitry Ustalov. 2022. TextGraphs 2022 shared task on natural language premise selection. In *Proceedings of TextGraphs-16: Graph-based Methods for Natural Language Processing*, pages 105–113, Gyeongju, Republic of Korea. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Sean Welleck, Jiacheng Liu, Ronan Le Bras, Hannaneh Hajishirzi, Yejin Choi, and Kyunghyun Cho. 2021. Naturalproofs: Mathematical theorem proving in natural language. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Sean Welleck, Peter West, Jize Cao, and Yejin Choi. 2022. Symbolic brittleness in sequence models: on systematic generalization in symbolic mathematics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8629–8637.

Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. 2019. A review of recurrent neural networks:

Algorithm 1 Premise Generation

$\mathcal{F} \leftarrow$ premise.free_symbols
$p \leftarrow \frac{1}{p_r} - 1$
for s in $\mathcal{F}$ **do**
  $m \leftarrow$ random.choice($[0] * p + [1]$)
  if $m = 0$ **then**
    $s' \leftarrow s$
  else
    $p_c \leftarrow \frac{1}{p_e} - 1$
    $m \leftarrow$ random.choice($[0] * p_c + [1]$)
    $c \leftarrow$ random.choice($\{2, ..., 9\}$)
    if $m = 0$ **then**
      if random.choice({0,1}) = 0 **then**
        $s' \leftarrow s \times c$
      else
        $s' \leftarrow \frac{s}{c}$
      end if
    else
      $c \leftarrow$ random.choice($\{2, ..., 9\}$)
      $s' \leftarrow s^c$
    end if
  end if
  premise $\leftarrow$ premise.subs(s, s')
end for
return premise

Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270.

A Data Generation

Algorithm 1 formalises the general data generation methodology adopted for generating premises with the SymPy[7] engine.

The following is an example of an entry in the dataset with both LaTex and Sympy surface form for representing expressions, considering *integration* and a single variable operand r. The same overall structure is adopted for the remaining operations and a larger vocabulary of variables:

- Premise:
  - Latex:

$$u + cos(log(-x + o))$$

  - SymPy:

$$Add(Symbol('u'),$$
$$cos(log(Add(Mul(Integer(-1),$$
$$Symbol('x')),$$
$$Symbol('o')))))$$

- Derivation (*integration*, r):
  - Latex:

$$ur + rcos(log(-x + o))$$

  - SymPy:

$$Add(Mul(Symbol('u'), Symbol('r')),$$
$$Mul(Symbol('r'), cos(log($$
$$Add(Mul(Integer(-1),$$
$$Symbol('x')), Symbol('o'))))))$$

B Dev Results

Table 3 reports the complete results on the dev set for different models and architectures with different embedding sizes.

[7] https://www.sympy.org/en/index.html

	MAP	Hit@1	Hit@3	MAP	Hit@1	Hit@3	Avg MAP	Embeddings Dim.	Model Size (MB)
Projection (One-hot)		**Cross-op.**			**Intra-op.**				
GCN	71.37	74.86	86.46	92.80	97.83	99.03	82.20	300	3.0
	73.44	**78.73**	89.50	92.72	97.70	**99.86**	83.08	512	7.9
	74.12	78.40	89.66	92.68	97.46	99.13	83.40	768	18.0
GraphSAGE	79.54	82.60	92.83	91.98	96.43	98.63	85.76	300	5.0
	81.57	84.90	95.00	92.81	97.50	99.20	87.19	512	14.0
	83.70	**88.56**	**96.73**	**93.00**	97.70	99.30	**88.35**	768	31.0
CNN	69.84	77.53	95.00	91.79	96.10	98.30	80.81	300	2.5
	66.94	74.46	93.40	92.06	96.80	98.40	79.50	512	4.6
	67.25	74.70	93.63	91.48	95.46	97.73	79.37	768	7.6
LSTM	69.31	72.06	89.00	92.93	**97.96**	99.46	81.12	300	6.6
	69.33	71.66	88.16	92.92	97.90	99.46	81.13	512	19.0
	70.84	72.60	90.10	92.89	97.76	99.30	81.86	768	42.0
Transformer	48.61	48.20	63.70	91.79	96.60	99.43	70.20	300	38.0
	46.29	43.70	61.30	91.72	96.46	99.16	69.01	512	75.0
	46.17	43.56	62.43	91.98	96.90	99.20	69.08	768	130.0
Projection (Dense)		**Cross-op.**			**Intra-op.**				
GCN	77.37	82.16	93.63	91.28	96.43	98.93	84.33	300	3.3
	77.89	83.46	92.70	92.45	97.33	98.90	85.17	512	8.9
	79.93	85.63	94.10	92.09	96.93	99.00	86.01	768	20.0
GraphSAGE	81.39	84.83	95.76	91.08	95.80	98.60	86.24	300	5.4
	80.73	84.06	94.20	92.36	97.10	98.86	86.54	512	15.0
	81.09	83.93	94.36	92.40	97.40	99.10	86.75	768	33.0
CNN	81.91	90.46	97.80	91.67	95.76	98.23	86.79	300	2.8
	82.70	**92.10**	**98.73**	91.89	95.93	98.33	**87.30**	512	5.6
	81.70	90.73	97.80	92.36	97.20	98.90	87.03	768	9.9
LSTM	71.96	74.93	89.03	92.26	96.93	99.26	82.11	300	7.0
	76.13	80.50	93.53	92.77	97.70	99.30	84.45	512	20.0
	76.40	80.03	93.23	**93.13**	**98.06**	**99.60**	84.76	768	44.0
Transformer	70.06	75.50	88.70	91.96	97.40	99.53	81.01	300	38.0
	69.59	73.63	87.20	92.20	97.70	99.53	80.89	512	76.0
	52.43	51.16	68.30	90.78	96.16	99.33	71.60	768	133.0
Translation		**Cross-op.**			**Intra-op.**				
GCN	80.16	89.50	96.63	83.90	86.83	94.16	82.03	300	3.0
	86.56	95.03	99.03	86.20	89.16	96.16	86.38	512	7.9
	86.72	95.53	99.30	87.85	91.40	96.50	87.29	768	18.0
GraphSAGE	84.94	92.93	98.10	84.13	86.00	93.40	84.53	300	5.0
	87.44	95.26	99.00	88.70	92.76	96.63	88.07	512	14.0
	88.39	96.13	99.16	90.35	94.20	97.86	89.37	768	31.0
CNN	84.42	95.30	98.93	89.22	93.66	97.66	86.81	300	2.5
	84.62	95.33	99.20	90.36	96.03	98.63	87.49	512	4.6
	86.99	96.76	**99.60**	87.18	93.46	96.90	87.08	768	7.7
LSTM	84.20	92.23	99.10	87.24	90.76	96.93	85.72	300	6.6
	86.98	94.36	99.06	88.81	93.70	98.00	87.89	512	19.0
	89.50	**97.13**	99.36	89.89	95.70	98.53	**89.70**	768	42.0
Transformer	84.17	94.90	98.83	90.35	96.23	99.40	87.26	300	38.0
	85.99	95.70	98.70	**91.86**	**97.83**	**99.60**	88.93	512	75.0
	86.04	95.33	98.53	91.27	97.03	99.20	88.65	768	130.0

Table 3: Overall performance of different neural encoders and methods (dev set) for jointly encoding multiple mathematical operations (i.e., integration, differentiation, addition, difference, multiplication, division).