A Symbolic Framework for Systematic Evaluation of Mathematical Reasoning with Transformers

A Symbolic Framework for Systematic Evaluation of Mathematical Reasoning with Transformers

Jordan Meadows[1], **Marco Valentino**[2], **Damien Teney**[2], **André Freitas**[1,2]
[1]University of Manchester, United Kingdom
[2]Idiap Research Institute, Switzerland
jordan.meadows@postgrad.manchester.ac.uk
{marco.valentino, damien.teney, andre.freitas}@idiap.ch

Abstract

Whether Transformers can learn to apply symbolic rules out-of-distribution is an open research question. In this paper, we devise a data generation method for producing intricate mathematical derivations, and systematically perturb them with respect to syntax, structure, and semantics. Our task-agnostic approach generates equations, annotations, and inter-equation dependencies, employing symbolic algebra for scalable data production and augmentation. We then instantiate a general experimental framework on next-equation prediction using 200K examples, and assess mathematical reasoning and generalisation capabilities of Transformer encoders. The experiments reveal that perturbations heavily affect performance and can reduce F1 scores of 97% to below 17%. This suggests that inference is dominated by surface-level patterns unrelated to a deeper understanding of mathematical operators, and underscores the importance of rigorous, large-scale evaluation frameworks for revealing fundamental limitations of existing models.

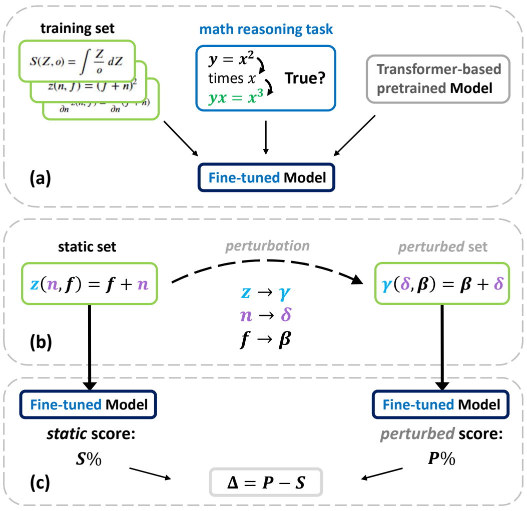

Figure 1: An overview of the proposed framework. We leverage computer algebra to generate large-scale training data for mathematical reasoning tasks **(a)** and apply systematic perturbations to examples from a static test set to form a perturbed test set **(b)**. The static evaluation scores are compared with scores on the perturbed set, given some metric, to determine model robustness and generalisation **(c)**.

1 Introduction

Systematicity and out-of-distribution generalisation in deep neural models, such as Transformers (Vaswani et al., 2017), are challenging yet crucial areas of research (Schlegel et al., 2023; Belinkov, 2022; Teney et al., 2020). Enhancing these capabilities could bolster model robustness (Kumar et al., 2020), facilitate transparent decision-making (Lee et al., 2022), and amplify complex reasoning abilities in science and mathematics (Frieder et al., 2023; Valentino et al., 2022b; Lewkowycz et al., 2022; Drori et al., 2022; Welleck et al., 2021). Various strategies have been proposed to evaluate model robustness, including direct input manipulation (Rozanova et al., 2023b; Stolfo et al., 2022; Nie et al., 2020; Kaushik et al., 2019) and probing on the internal representation (Rozanova et al., 2023a; Ravichander et al., 2021; Elazar et al., 2021; Veitch et al., 2020).

This paper considers input interventions through syntactic, structural and semantic perturbations to mathematical text. Current interventional approaches are challenged by the difficulty of isolating confounding factors, and formalising the expected causal mechanisms that underpin models' predictions (Rozanova et al., 2023b; Stolfo et al., 2022; Ribeiro et al., 2020; Kaushik et al., 2019). These hurdles impact the scope and reliability of causality and robustness studies (Pearl, 2009; Shreya et al., 2022).

To tackle existing limitations, we leverage the rich environment of symbolic algebra to design a task-agnostic and systematic evaluation framework. Strict symbolic rules offer a systematic approach to perturbing mathematical reasoning and evaluating out-of-distribution generalisation of neural models. This allows us to perturb multiple elements of math reasoning, covering structural, semantic, and syntactic aspects across diverse mathematical subdomains, extending beyond the limited interventional scope of previous works (Stolfo et al., 2022; Patel et al., 2021; Ribeiro et al., 2020; Kaushik et al., 2019; Yao et al., 2021).

Additionally, we address an impending data scarcity problem, where high-quality data is forecast to be outpaced by the training needs of models within the decade (Villalobos et al., 2022). Symbolic engines facilitate the generation of annotated mathematical reasoning, which allows the construction of high-quality datasets for various tasks. We combine (18) symbolic operators with simple rules that guide exploration of equational state spaces and generate derivations, then perturb and adapt them for specific entailment tasks. These are sequence classification tasks involving next-equation prediction that focus on operator usage in annotated derivations, or the direct integration or differentiation of expressions.

To validate our framework, we test a canon of Transformer encoders used in mathematical language processing (Li et al., 2023; McNichols et al., 2023; Zhong et al., 2022; Meadows and Freitas, 2022) to determine their capacity for learning how operators work out-of-distribution, and to abstract fundamental properties impacting their ability to generalise.

To summarise, the paper offers the following contributions:

1. An approach to generating annotated equational derivations of controllable complexity levels, involving premise equation generation (Algorithm 1) and the sequential application of operators to premises (Algorithm 2).

2. A systematic and scalable methodology to perturb various aspects of mathematical language including structure, syntax, and semantics.

3. Example instantiations of the framework involving sequence classification tasks based on next-equation prediction. The generated datasets include static and perturbed derivations totalling 200K task-specific examples.

4. An extensive experimental framework for training models on mathematical reasoning tasks and evaluating their robustness, including dataset generation, systematic perturbation, training, and evaluation (Fig. 1).

5. An empirical evaluation of Transformer encoders used in mathematical language processing. Our results suggest that models are not predominantly learning abstract rules in this context, and inference heavily depends on superficial textual patterns unrelated to deeper mathematical understanding.

To the best of our knowledge, we are the first to propose a general algebraic solver-based framework for producing large-scale and systematic evaluation benchmarks for mathematical reasoning with Transformers. We release the data generation algorithms and the complete datasets adopted for the experiments[1].

2 Related Work

This work meets at the intersection of computer algebra, mathematical reasoning with Transformer-based models, and evaluation frameworks. We briefly describe the landscape for each domain.

Computer algebra. SymPy (Meurer et al., 2017) is a computer algebra system used in conjunction with a number of language processing methods. For example, Chen et al. (2022) solve numerical reasoning tasks including simple math elements such as numbers, by chain-of-thought prompting language models to generate SymPy solvable code. Mandlecha et al. (2022) use SymPy to generate data for answering questions ranging from arithmetic to calculus, tasks, without testing for generalisability. Hu and Yu (2022) solve a similar array of problems from a large-scale dataset (Saxton et al., 2019), and test for generalisability to an extrapolation set of problems. Drori et al. (2022) fine-tune the decoder model, Codex (Chen et al., 2021), on a dataset of questions from MIT's university-level mathematics courses, generating SymPy solution code. Meadows and Freitas (2021) incorporate computer algebra with a basic heuristic search to reconstruct derivations from condensed matter research (Mann et al., 2018).

Reasoning with mathematical language. Since early reasoning work with Transformers (Saxton et al., 2019; Clark et al., 2020; Rabe et al., 2020), they have revolutionised multiple subdomains of

[1]`https://github.com/anonymous/TBA`

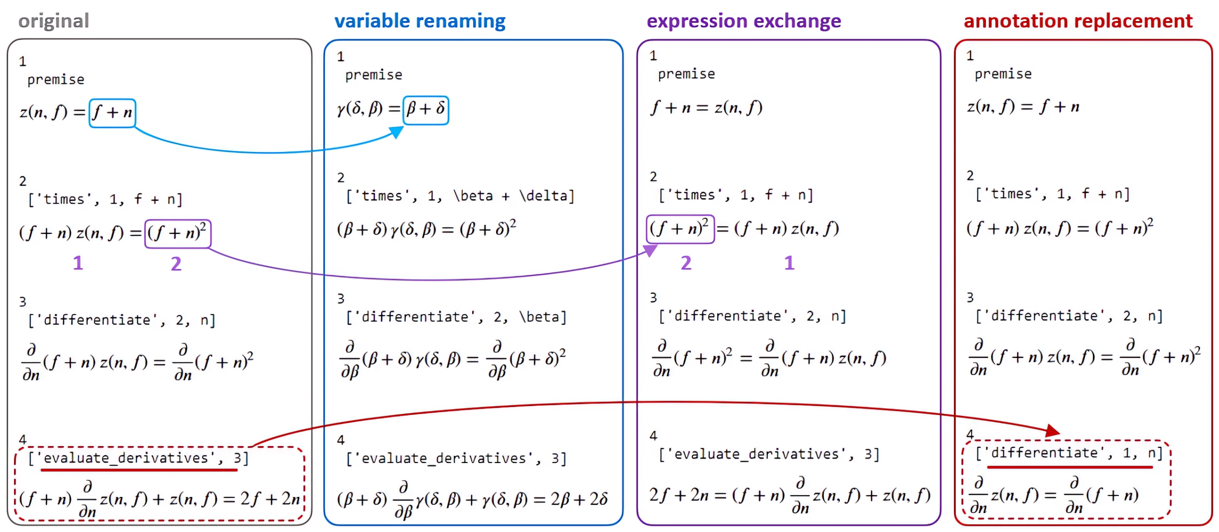
original
variable renaming
expression exchange
annotation replacement
1 premise
$z(n, f) = \boxed{f + n}$
2 ['times', 1, f + n]
$(f+n)\, z(n,f) = \boxed{(f+n)^2}$
1 2
3 ['differentiate', 2, n]
$\frac{\partial}{\partial n}(f+n)\, z(n,f) = \frac{\partial}{\partial n}(f+n)^2$
4 ['evaluate_derivatives', 3]
$(f+n)\frac{\partial}{\partial n} z(n,f) + z(n,f) = 2f + 2n$
1 premise
$\gamma(\delta, \beta) = \boxed{\beta + \delta}$
2 ['times', 1, \beta + \delta]
$(\beta+\delta)\, \gamma(\delta,\beta) = (\beta+\delta)^2$
3 ['differentiate', 2, \beta]
$\frac{\partial}{\partial \beta}(\beta+\delta)\, \gamma(\delta,\beta) = \frac{\partial}{\partial \beta}(\beta+\delta)^2$
4 ['evaluate_derivatives', 3]
$(\beta+\delta)\frac{\partial}{\partial \beta}\gamma(\delta,\beta) + \gamma(\delta,\beta) = 2\beta + 2\delta$
1 premise
$f + n = z(n,f)$
2 ['times', 1, f + n]
$\boxed{(f+n)^2} = (f+n)\, z(n,f)$
2 1
3 ['differentiate', 2, n]
$\frac{\partial}{\partial n}(f+n)^2 = \frac{\partial}{\partial n}(f+n)\, z(n,f)$
4 ['evaluate_derivatives', 3]
$2f + 2n = (f+n)\frac{\partial}{\partial n} z(n,f) + z(n,f)$
1 premise
$z(n,f) = f + n$
2 ['times', 1, f + n]
$(f+n)\, z(n,f) = (f+n)^2$
3 ['differentiate', 2, n]
$\frac{\partial}{\partial n}(f+n)\, z(n,f) = \frac{\partial}{\partial n}(f+n)^2$
4 ['differentiate', 1, n]
$\frac{\partial}{\partial n} z(n,f) = \frac{\partial}{\partial n}(f+n)$

Figure 2: Example perturbations applied to a derivation using computer algebra. Given the *next-equation prediction* task chosen, annotation replacement (red) is *semantics-altering*, while the others are *semantics-preserving*. Variable Renaming (**VR**) involves replacing symbols with out-of-distribution Greek letters. Expression Exchange (**EE**) swaps expressions either side of an equality symbol. Annotation Replacement (**AR**) selects an alternative final annotation that leads to an alternative final equation.

mathematical language processing (Meadows and Freitas, 2022) and are responsible for headlining results (Lewkowycz et al., 2022; Drori et al., 2022). Transformer encoder models obtain state-of-the-art performance in variable typing (Ferreira et al., 2022; Lai et al., 2022), formula search (Zhong et al., 2022; Peng et al., 2021), natural language premise selection (Valentino et al., 2022a; Tran et al., 2022), and retrieval-based math question answering (Reusch et al., 2022; Novotný and Štefánik, 2022), among others.

Data-augmentation and evaluation frameworks. In particular, Stolfo et al. (2022) perturb elements of math word problems (Liang et al., 2022) such as numerical operands of implicit arithmetic operations, and natural language. Inspired by related work (Patel et al., 2021; Ribeiro et al., 2020), they apply a causal analysis (Christiansen et al., 2021) to determine the effect of do-interventions (Pearl, 2022). Their data-augmentation approach is limited to one or two task-dependent interventions. Our approach is task-agnostic, systematic, and scalable, and allows for complex changes to mathematical elements such as operators, variables, expressions, and equations.

3 Generating and Perturbing Derivations with Computer Algebra

We describe the general framework for generating derivations from a vocabulary of symbols and a set of operators. The operators include addition, subtraction, multiplication, division, exponentiation, cos, sin, log, exp, operations for setting up derivatives and integrals and evaluating them, expression substitutions, and operations for defining premises. An example of a generated derivation is given in Fig. 3.

3.1 Premise Generation

Derivations rely on premise equations. Operations are applied to premises to generate new equations, as shown in Fig. 3. The first step in Fig. 3 is the premise equation (1). We outline our approach to generating premises in this subsection, using a vocabulary, and a set of operations defined within a computer algebra system.

The vocabulary includes uppercase and lowercase English characters, excluding {i, e, d, O}, due to their connection with math concepts. Operators are classified by their arity. For example, the symbols Z and o are sampled from the vocabulary and used as operands for the 2-arity operator "divide". Then, Z is sampled from the vocabulary as an operand for the 1-arity operator "integrate". This expression becomes $\int \frac{Z}{o} dZ$, and consists of the free symbols Z and o. This is the RHS of the premise equation.

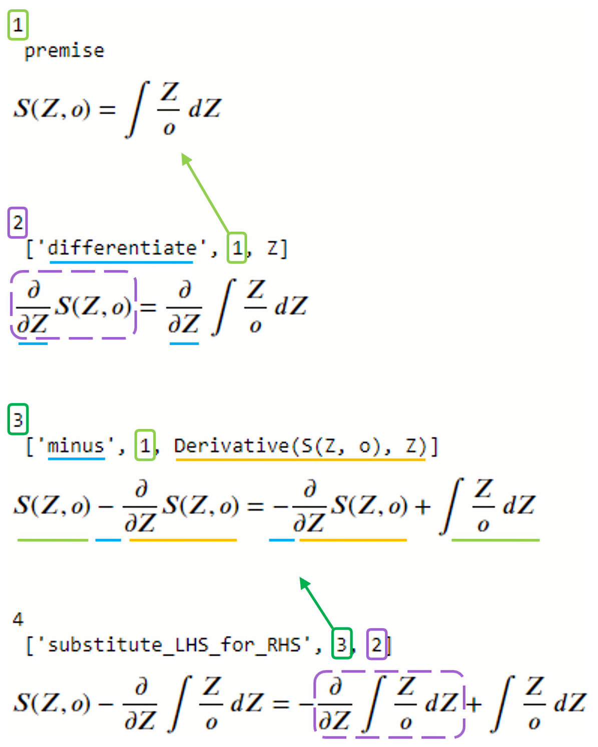
1
premise
$S(Z,o) = \int \frac{Z}{o}\, dZ$
2
['differentiate', 1, Z]
$\frac{\partial}{\partial Z} S(Z,o) = \frac{\partial}{\partial Z} \int \frac{Z}{o}\, dZ$
3
['minus', 1, Derivative(S(Z, o), Z)]
$S(Z,o) - \frac{\partial}{\partial Z} S(Z,o) = -\frac{\partial}{\partial Z} S(Z,o) + \int \frac{Z}{o}\, dZ$
4
['substitute_LHS_for_RHS', 3, 2]
$S(Z,o) - \frac{\partial}{\partial Z} \int \frac{Z}{o}\, dZ = -\frac{\partial}{\partial Z} \int \frac{Z}{o}\, dZ + \int \frac{Z}{o}\, dZ$

Figure 3: A generated derivation using the proposed computer algebra system. Colours highlight the dependencies between different reasoning steps.

To form the LHS, a function symbol is sampled from the vocabulary, in this case S, and the two free symbols are assigned as variables. The LHS and RHS are themselves inputted as arguments of an equation operation, and the premise (1) is obtained. A formal description is given by Algorithm 1 in Appendix B.

3.2 Equational Reasoning

Operations accept premise equations and sampled math elements, and generate new equations. All generated equations in a derivation may be used to derive the next equation. We describe this formally in Algorithm 2, including a description of how we sample from equation distributions to emulate human-like derivations, in Appendix C. Operators are classified by their arity $\in [0, 2]$, and are naturally applied to each side of an equation.
For example, starting from the premise in Fig. 3, given by

$$S(Z,o) = \int \frac{Z}{o} dZ, \quad (1)$$

the 2-arity class is selected, and "differentiate" is chosen from operators matching that arity. All valid expressions and variables are sampled from, and the algorithm selects Z. The annotation ['differentiate', 1, Z], means the operator was applied to operand equation (1), with the second operand Z, and is applied to both LHS and RHS of (1) to give

$$\frac{\partial}{\partial Z} S(Z,o) = \frac{\partial}{\partial Z} \int \frac{Z}{o} dZ. \quad (2)$$

The notation ['minus', 1, Derivative(S(Z,o), Z)] means that an 2-arity operation was selected, the operator "minus" was chosen, and the LHS of (2) was selected as the second operand. This operand is subtracted from both sides of first operand equation (1), to give

$$S(Z,o) - \frac{\partial}{\partial Z} S(Z,o) = -\frac{\partial}{\partial Z} S(Z,o) + \int \frac{Z}{o} dZ. \quad (3)$$

The final step, annotated by ['substitute_LHS_for_RHS', 3, 2], means the substitution function was chosen, and equations (3) and (2) are the first and second operands. In this case, the LHS of (2) was identified within (3), and substituted for the RHS of (2), to give:

$$S(Z,o) - \frac{\partial}{\partial Z} S(Z,o) = -\frac{\partial}{\partial Z} \int \frac{Z}{o} dZ + \int \frac{Z}{o} dZ. \quad (4)$$

This procedure, formalised in Algorithm 2, allows for a systematic and scalable generation of equational derivations, with grounded symbolic properties.

3.3 Perturbations

A perturbation targets a single aspect of a derivation (such as variable names), and either preserves the semantics of the original derivation or alters it in a specific and controllable way.

Semantics-preserving. An input derivation sequence, such as Fig. 4(a), is associated with a label determining its truth value (in the current classification context). Semantics-preserving perturbations generate examples that preserve the semantic link between sequence and label, and the given sequence is still True after the change. Fig. 2 describes *variable renaming* (**VR**) and *expression exchange* (**EE**) perturbations of this type. Variable renaming involves sampling variables from a different vocabulary to that of the training data. We sample from a set of ten Greek symbols and replace English variables using SymPy substitution operations. The EE perturbation generates reordered equations, where the only change is the position of top-level expressions either side of the equality operator. We also necessarily reverse any asymmetric annotations with respect to LHS or RHS of equations. For our set of operations, this means replacing the substitution function, "substitute_LHS_for_RHS", with its RHS equivalent, and vice versa. Due to a premise always featuring a function on the LHS, it never sees EE examples during training, and due to the Greek symbols, VR examples are also out-of-distribution. We also apply variable renaming and *equality conversion* perturbations to examples from the *direct calculus* task variation. As described in Fig. 4(b), a differentiation example input may be x^2 [SEP] x [SEP] $2x$. *Equality conversion* (**EC**) involves converting the valid expressions into equations, by sampling the LHS symbol from unused symbols. The EC perturbed example in this case may be $y = x^2$ [SEP] x [SEP] $\frac{d}{dx}y = 2x$. For integration, the first expression becomes the equation containing the differential operator. This perturbation is the simplest possible introduction of an equality operator while maintaining mathematical correctness.

Semantics-altering. A perturbation of this type alters the semantic link between the sequence and label. A perturbed sequence is now False if it was previously True. As in Fig 4(a), an input consists of previous derivation steps, an annotation associated with an operator used to generate the final equation, and the *candidate* final equation. This equation is either the ground truth or a negative example, and the input is paired with a label that reflects the overall sequence coherence. Both true and false versions of a sequence are seen during training, that differ only by the final equation. To alter the semantic link between sequence and label meaningfully, the *annotation replacement* (**AR**) perturbation replaces a final annotation in a way that the *negative example* corresponds to the *positive label*, and vice versa. The classification labels are then swapped to reflect the change. This perturbation is possible because we generate negative examples by applying alternative final operations to equations using computer algebra. We do not apply the AR perturbation in direct calculus, because negatives are generated differently for that task. As mentioned in Section 4, negatives for direct calculus are selected by ranking generated premise expression lists with a string metric. Fig.2 and Fig. 1(b) display the effect of perturbations.

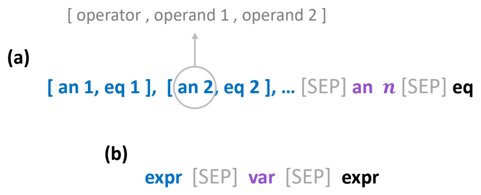
[operator , operand 1 , operand 2]
(a)
[an 1, eq 1], [an 2, eq 2], ... [SEP] an n [SEP] eq
(b)
expr [SEP] var [SEP] expr

Figure 4: Two variations of encoder input for the next-equation prediction task: (a) structured derivations, and (b) direct calculus.

4 Framework Instantiation

4.1 Next-Equation Prediction

We instantiate the general framework described in Section 3 for evaluation, formalising two sequence classification tasks as a next-equation prediction.

Next-equation prediction is a binary sequence classification task. Given all equations and annotations in the derivation so far, including the final annotation that describes how to construct the ground truth final equation, a candidate equation is paired with the prior context, and the model learns whether the reasoning entails this equation. We generate two variations of the task: the above describes the *structured derivations* variation that relies on Algorithm 2, while *direct calculus* involves single-step differentiation and integration of premise expressions generated with Algorithm 1.

Structured derivations. Fig. 4(a) describes the input format for generated derivations. A step consists of an equation and an annotation, as described in Fig. 3. An annotation is a list comprising an operator name and its operands. Each step [an, eq] is linearised and comma separated, up to the final step. The final step annotation is separated from the derivation, and the final equation is replaced with a negative example equation, or left unchanged. All three input components are [SEP] separated as in Fig. 4(a). Negative examples are generated by applying a different operation to a previous equation. Any previous equations may be used to generate the final equation, meaning there are long-range dependencies. Mathematically, the model should learn the necessary equation dependencies required to form the final equation, and how to apply the correct operator (guided by the final annotation).

Direct calculus. In this task we emphasise a single-step evaluation of derivatives and integrals. Fig. 4(b) describes the input. A premise expression containing *at least two* variables is generated, a variable is randomly selected from the premise,

Task	Train	Dev	Test
Structured derivations			
2-steps	20K	5K	4K
3-steps	20K	5K	4K
4-steps	20K	5K	4K
direct calculus			
integration	32K	8K	4K
differentiation	32K	8K	4K

Table 1: The number of examples considered by models during training, development, and evaluation.

and the resulting expression after differentiating or integrating with respect to that variable is the ground truth. This positive example is either replaced with a negative example, or not. A classifier infers if the reasoning context generated the final expression. Negative examples are generated by selecting from a list of alternative premise expressions. This list includes the result of differentiating/integrating the expression with respect to other variables in the expression, and differentiating/integrating other randomly generated expressions comprised of the same symbols. The list of expressions are then ranked in terms of their Damerau-Levenshtein distance (Zhao and Sahni, 2019) from the ground truth (Meadows and Freitas, 2021). For example, the expression $-T + sin(U)$ is differentiated with respect to T to give -1. The corresponding negative example is 1. The expression, variable, and candidate expression are [SEP] separated upon input to the model, as shown in Fig. 4(b).

4.2 Data Generation

We construct datasets that allow for derivation reconstruction within the computer algebra system, such that they may be further perturbed or extended. The derivations themselves are task-agnostic, but we include negative equations from the current task for reproducibility. A single entry consists of the reasoning sequence up to the final expression or equation (see Fig. 4). This sequence is grouped with both the correct final equation and negatives, and is stored in both LaTeX and SymPy-interpretable language (Meurer et al., 2017). Before a model encounters an example, it is processed into two sequences: one including the positive equation, one including the negative, along with their classification labels. The number of examples seen by models is displayed in Table 1. Perturbations are applied to the test set and generate an equal number of perturbed examples. The *structured derivation* datasets include 20k training, 5k development, and 4k evaluation examples. *Direct calculus* includes 32k training, 5k development, and 4k evaluation examples.

Generalisation to extrapolation examples. A model that can sufficiently generalise should be able to solve mathematically less complex versions of problems encountered during training. The structured derivations task is split into a further three variants: those considering 2, 3, and 4 step derivations. 4-step derivations are intuitively the hardest, as the static evaluation supports, and 2-step derivations are the easiest. To test for generalisability to an extrapolation set. We evaluate models trained on derivations with *higher* step counts, on derivations with *lower* step counts. This is represented in the **s - 1** and **s - 2** columns in Table 2, where **s** is the number of steps the model was trained on. Models solving the direct calculus task are trained/evaluated on examples comprising *at least two* variables, *e.g.,* $\cos(ax) - z$. We generate a set of easier calculus problems with 1.5k examples that consist of only one variable, *e.g.,* $\cos(x)$.

5 Evaluation

As described in Fig. 1, we first evaluate models on a static set, apply perturbations to the static set examples, evaluate models on the perturbed sets, and compare the difference between scores (accuracy and F1). In addition, we evaluate on *extrapolation* examples described in the previous section, including derivations consisting of less steps, and direct calculus examples consisting of functions of single variables. Table 2 *(structured derivations)* and Table 3 *(direct calculus)* display results from next-equation prediction experiments.

5.1 Results

Models fail to generalise. For *structured derivations*, models average 80% F1 over all static derivation lengths, and decreases due to perturbations average 10% (VR), 11% (EE), and 16% (AR). This is at most 4% above F1 majority baseline. BERT-uncased and SciBERT-cased fine-tuned on 2-step derivations are exceptions, but the 13 other models are sensitive to at least one perturbation. All models tested do not generalise to *less* derivation steps, reaching as low as 11% F1.

	Static		**VR**		**EE**		**AR**		**s - 1**		**s - 2**	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
BERT-base-uncased (**s=2**)	0.877	0.889	0.870	**0.881**	0.870	0.880	0.875	0.887	-	-	-	-
BERT-base-uncased (**s=3**)	0.789	0.787	0.719	0.710	0.691	0.660	0.537	0.506	0.684	0.690	-	-
BERT-base-uncased (**s=4**)	0.588	0.636	0.550	0.603	0.564	0.603	0.424	0.481	0.657	0.622	0.528	0.298
BERT-base-cased (**s=2**)	0.872	0.885	0.819	0.832	0.853	0.861	0.855	0.872	-	-	-	-
BERT-base-cased (**s=3**)	0.782	0.773	0.688	0.645	0.650	0.589	0.545	0.496	0.546	0.305	-	-
BERT-base-cased (**s=4**)	0.668	0.717	0.585	0.615	0.626	***0.672***	0.433	0.531	0.719	0.739	0.543	0.218
MathBERT (**s=2**)	0.832	0.820	0.762	0.706	0.790	0.757	0.785	0.760	-	-	-	-
MathBERT (**s=3**)	0.842	0.839	0.691	0.645	0.633	0.522	0.663	0.640	0.674	0.587	-	-
MathBERT (**s=4**)	0.671	0.684	0.595	0.526	0.623	0.621	0.485	0.479	0.686	0.680	0.518	0.290
SciBERT-uncased (**s=2**)	0.925	0.926	0.729	0.704	0.868	0.861	0.900	0.902	-	-	-	-
SciBERT-uncased (**s=3**)	0.889	***0.894***	0.821	***0.819***	0.703	***0.664***	0.709	***0.722***	0.806	***0.818***	-	-
SciBERT-uncased (**s=4**)	0.763	***0.765***	0.695	***0.668***	0.686	0.659	0.607	***0.596***	0.769	***0.779***	0.593	***0.574***
SciBERT-cased (**s=2**)	0.926	**0.931**	0.853	0.871	0.898	**0.902**	0.910	**0.917**	-	-	-	-
SciBERT-cased (**s=3**)	0.772	0.724	0.727	0.672	0.610	0.441	0.508	0.295	0.529	0.128	-	-
SciBERT-cased (**s=4**)	0.710	0.709	0.651	0.646	0.666	0.654	0.470	0.429	0.779	0.749	0.527	0.110
Average (**s=2**)	0.886	0.890	0.807	0.799	0.856	0.853	0.865	0.868	-	-	-	-
Average (**s=3**)	0.815	0.803	0.729	0.698	0.657	0.575	0.592	0.532	-	-	-	-
Average (**s=4**)	0.680	0.702	0.615	0.612	0.633	0.642	0.484	0.503	-	-	-	-
Average over all steps	0.794	0.798	0.717	0.703	0.715	0.690	0.647	0.634	-	-	-	-

Table 2: Model performance on the structured derivations variation of the next-equation prediction task. The **Static** column shows scores on data that is unperturbed with respect to the training data. **VR** (variable renaming) shows scores after renaming variables as Greek letters. **EE** (expression exchange) shows scores after swapping expressions either side of equality symbols in equations. **AR** (annotation replacement) shows scores after deriving an alternative final equation. **s - n** shows scores after evaluating on derivations with n less steps than training derivations (for $n \in \{1, 2\}$). Bold numbers denote highest F1 scores for **2-step** derivations. Bold italic numbers denote highest ***3-step*** scores. Bold, italic, and underlined numbers denote highest ***4-step*** scores.

In *direct calculus* static scores average 90% and perturbations decrease this by 17% (VR) and 33% for Equation Conversion (EC). All 10 fine-tuned models completely fail to generalise to perturbations and easier examples, with 97% F1 scores repeatedly dropping below 17%.

Entailment pre-training improves generalisability. BERT (Devlin et al., 2018) was trained on masked language modelling (MLM) and next sentence prediction (NSP) objectives. SciBERT (Beltagy et al., 2019) was fine-tuned with scientific papers on MLM and NSP. MathBERT (Peng et al., 2021) was fine-tuned with scientific papers on standard and structural MLM, and a context correspondence objective (related to NSP). Fine-tuning generally overwrites representations learned from previous tasks (Mosbach et al., 2020), so it is likely that MathBERT as forgotten those associated with NSP (compared to BERT or SciBERT). The context correspondence objective used to train MathBERT learns to pair the *description of an equation* with the equation itself. In contrast, NSP involves recognising consecutive sentences which – particularly in scientific text – better teaches *logical entailment*. Next-equation prediction considers if *context entails the equation in an argumentative sense*, rather than a descriptive sense. We therefore attribute generalisability failures of MathBERT to insufficient entailment pre-training. It has struggled with entailment before (Meadows et al., 2022).

Learning formula structure instead of entailment does not necessitate structural perturbation invariance. Expression Exchange (EE) and Equation Conversion (EC) involve perturbing implicit tree-structures of equations, such as operator trees (Mansouri et al., 2019). Despite MathBERT using a dedicated pre-training objective for learning equation tree structure, it is not more robust to structural perturbations than other models.

5.2 Qualitative Analysis

We consider (uncased) models trained on 3-step derivations. This number of steps closely reflects the average results over all step counts in Table 2. The **All** (perfect generalisation) and **Not P** (complete generalisation failure) columns of Table 4 reinforce the relative generalisability gap

	Static		VR		EC		Easy	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1
BERT-base-uncased (**int**)	0.900	0.907	0.688	0.704	0.751	0.780	0.627	**0.729**
BERT-base-uncased (**diff**)	0.759	0.803	0.649	0.733	0.622	***0.698***	0.551	0.691
BERT-base-cased (**int**)	0.930	0.934	0.716	**0.777**	0.852	**0.867**	0.638	0.718
BERT-base-cased (**diff**)	0.742	0.779	0.642	0.724	0.603	0.649	0.567	***0.696***
MathBERT (**int**)	0.922	0.923	0.744	0.758	0.744	0.718	0.586	0.686
MathBERT (**diff**)	0.847	0.859	0.597	0.481	0.584	0.473	0.561	0.500
SciBERT-uncased (**int**)	0.968	0.968	0.656	0.744	0.541	0.158	0.626	0.711
SciBERT-uncased (**diff**)	0.918	0.923	0.726	***0.765***	0.668	0.581	0.552	0.678
SciBERT-cased (**int**)	0.971	**0.972**	0.681	0.758	0.542	0.170	0.580	0.671
SciBERT-cased (**diff**)	0.923	***0.927***	0.709	0.765	0.654	0.546	0.615	0.723
Average (**int**)	0.938	0.932	0.697	0.748	0.686	0.537	0.611	0.703
Average (**diff**)	0.838	0.858	0.665	0.694	0.626	0.589	0.569	0.658
Average over both tasks	0.888	0.895	0.681	0.721	0.656	0.563	0.590	0.681

Table 3: Model performance on the direct calculus variation of next-equation prediction. The **Static** column shows scores on unperturbed data. **VR** (variable renaming) shows scores after renaming variables as Greek letters. **EC** (equation conversion) shows scores after rewriting expressions as equivalent equations that preserve the semantics of the reasoning. **Easy** shows scores on simpler examples that consist of only one variable (in comparison to at least two variables in the static set). Bold numbers denote highest F1 scores for **integration** derivations. Bold italic denotes highest ***differentiation*** scores.

	Static $\pm$	**All**	**Not P**
BERT	62.3 $R \int_E \partial_E$	7.4 $\partial_E \int -$	5.3 $S_L \int_E R$
SciBERT	79.6 $R \int_E \partial_E$	21.3 $\int \partial_E \cos$	1.6 $R\, X^O \times$
MathBERT	70.3 $R \int_E \int$	7.8 $\int \cos \sin$	9.3 $R\, \partial_E \int_E$

Table 4: **Static** $\pm$ is the rate at which positive and associated negative *unperturbed* sequences are *both* correctly classified. **All** (perfect generalisation) is the percentage of examples where the static and perturbed (positive and negative) sequences are correctly classified. **Not P** (complete failure to generalise) is percentage of examples where only the static positive sequences are classified correctly, while all perturbed positive sequences are incorrect. Symbols correspond to the top three most frequent (final) operators in each unperturbed sequence, where frequency is normalized with respect to operator frequency in the static set. R is a premise renaming operator. $\int$ and ∂ are integration and differentation operators. $\int_E$ and ∂_E are respective evaluation operators. X^O is exponentiation, $\times$ is multiplication, $-$ is subtraction, and S_L is LHS substitution.

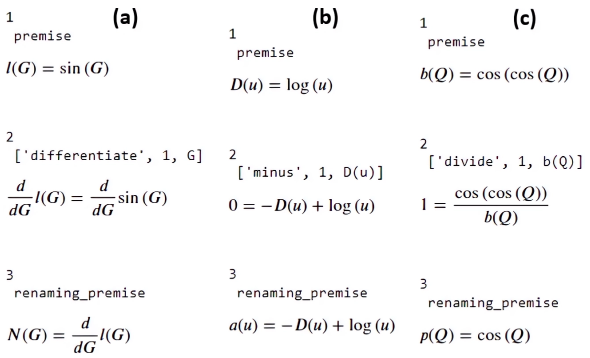

Figure 5: Three examples of the total 15, where both SciBERT and MathBERT correctly classify *unperturbed* examples (as shown), but incorrectly classify all perturbed examples.

between SciBERT and MathBERT, despite both being trained on scientific corpora, and display the top three operators by normalised frequency per generalisation category.

Generalisation failure depends on the unpredictability of an operator. For examples where models perfectly generalise, the operator responsible for setting up an integral (without evaluating it) is most common. This is likely because it involves prepending a unique text span "\int" to expressions either side of equations, which is easy to identify. Models generalise well to cos, sin, exp, and log operators, likely due to their similarly predictable effect on equations associated with regular text spans.

To highlight that it is likely the relative unpredictability of an operator's effect on text that leads to generalisation failure, we analyse the set of examples where *both* SciBERT and MathBERT correctly classify unperturbed sequences, but misclassify *all* perturbed sequences. Three examples are displayed in Fig. 5. The *renaming premise* operation is overwhelmingly frequent. It takes a random previously defined expression as the RHS, and defines a new function as the LHS. It does not necessarily depend on a single previous step and is non-deterministic due to random sampling of the RHS, yet it can never generate more complex equations than those previously derived (unlike all other operators).

6 Conclusion

We propose the use of mathematical reasoning generation algorithms to generate synthetic data for training language models to derive equations. In this case, we fine-tune Transformer encoder models to classify correct use of operators when deriving equations, through next-equation prediction tasks. Models obtain high scores on unperturbed test sets, but largely fail to generalise to perturbed test sets systematically generated with computer algebra. This instantiation of the framework suggests models are relying on textual patterns mostly unrelated to any deeper mathematical understanding of the operators. We explore the relationship between generalisation failure and operator usage, and determine that operators with a less predictable (and identifiable) effect on the surface form of equations reliably leads to generalisation failure, even if the mathematics is not necessarily more difficult. Although models that incorporate formula structure are strong in many tasks (Zhong et al., 2022; Meadows and Freitas, 2022; Peng et al., 2021), we suggest this should not overwrite knowledge that is crucial to the application at hand, such as textual entailment in next-equation prediction. Future research may explore the flexibility of the proposed symbolic framework to instantiate novel math reasoning tasks, investigate the systematic behaviour of larger language models (Chung et al., 2022; Brown et al., 2020), and incorporate causal analysis (Stolfo et al., 2022).

7 Limitations

Overall ethical impact. This work explores a systematic way to elicit the mathematical/symbolic inference properties of Transformer-based models in mathematical language processing tasks. As such, it contributes in the direction of a critique of the true reasoning capabilities and biases of these models.

Derivation generation. There are irrelevant steps in some longer derivations, such as applying an operation to an equation, but not using the result. This should not affect results as the final equation always depends on a previous equation, except when it is a renaming premise. This error is likely due to incorrect subderivation extraction, and will be improved.

Perturbations. Perturbations should only change a single aspect of the input, controlling for all other casual aspects. The variable renaming perturbation should only replace variables. However, the difference between the use of Greek and English alphabet, is the wordpiece tokenizer splits *e.g.,*β into two tokens, meaning attention scores are calculated between them. This does not occur for English characters. Also, due to SymPy ordering limitations, a change in notation may change the ordering of commutative variables within expressions. Therefore, this implementation does not lead to an pure perturbation that only changes a single mathematical property, and further artefacts are present within the evaluation.

Integration. SymPy does not generate integration constants. Although we account for this within derivation generation, we choose not to for direct calculus of integrals. Also, many integrals are evaluated to be case-based expressions, including value inequalities. We omit these examples for a closer comparison with differentiation, and for better compatibility with the perturbations.

Acknowledgements

This work was partially funded by the Swiss National Science Foundation (SNSF) project NeuMath (200021_204617).

References

Yonatan Belinkov. 2022. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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*.

Rune Christiansen, Niklas Pfister, Martin Emil Jakobsen, Nicola Gnecco, and Jonas Peters. 2021. A causal framework for distribution generalization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6614–6630.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. *arXiv preprint arXiv:2002.05867*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, Roman Wang, Nikhil Singh, Taylor L. Patti, Jayson Lynch, Avi Shporer, Nakul Verma, Eugene Wu, and Gilbert Strang. 2022. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32).

Yanai Elazar, Shauli Ravfogel, Alon Jacovi, and Yoav Goldberg. 2021. Amnesic probing: Behavioral explanation with amnesic counterfactuals. *Transactions of the Association for Computational Linguistics*, 9:160–175.

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.

Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Christian Petersen, Alexis Chevalier, and Julius Berner. 2023. Mathematical capabilities of chatgpt. *arXiv preprint arXiv:2301.13867*.

Yangyang Hu and Yang Yu. 2022. Enhancing neural mathematical reasoning by abductive combination with symbolic library. *arXiv preprint arXiv:2203.14487*.

Divyansh Kaushik, Eduard Hovy, and Zachary C Lipton. 2019. Learning the difference that makes a difference with counterfactually-augmented data. *arXiv preprint arXiv:1909.12434*.

Ram Shankar Siva Kumar, Magnus Nyström, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissoneru, Matt Swann, and Sharon Xia. 2020. Adversarial machine learning-industry perspectives. In *2020 IEEE security and privacy workshops (SPW)*, pages 69–75. IEEE.

Viet Lai, Amir Pouran Ben Veyseh, Franck Dernoncourt, and Thien Nguyen. 2022. Semeval 2022 task 12: Symlink-linking mathematical symbols to their descriptions. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pages 1671–1678.

Rebecca J Lee, Oskar Wysocki, Cong Zhou, Rohan Shotton, Ann Tivey, Louise Lever, Joshua Woodcock, Laurence Albiges, Angelos Angelakas, Dirk Arnold, et al. 2022. Establishment of coronet, covid-19 risk in oncology evaluation tool, to identify patients with cancer at low versus high risk of severe complications of covid-19 disease on presentation to hospital. *JCO clinical cancer informatics*, 6:e2100177.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*.

Weixian Waylon Li, Yftah Ziser, Maximin Coavoux, and Shay B Cohen. 2023. Bert is not the count: Learning to match mathematical statements with proofs. *arXiv preprint arXiv:2302.09350*.

Zhenwen Liang, Jipeng Zhang, Lei Wang, Wei Qin, Yunshi Lan, Jie Shao, and Xiangliang Zhang. 2022. Mwp-bert: Numeracy-augmented pre-training for math word problem solving. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 997–1009.

Pratik Mandlecha, Snehith Kumar Chatakonda, Neeraj Kollepara, and Pawan Kumar. 2022. Hybrid tokenization and datasets for solving mathematics and science problems using transformers. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 289–297. SIAM.

Charlie-Ray Mann, Thomas J Sturges, Guillaume Weick, William L Barnes, and Eros Mariani. 2018. Manipulating type-i and type-ii dirac polaritons in cavity-embedded honeycomb metasurfaces. *Nature communications*, 9(1):1–11.

Behrooz Mansouri, Shaurya Rohatgi, Douglas W Oard, Jian Wu, C Lee Giles, and Richard Zanibbi. 2019. Tangent-cft: An embedding model for mathematical formulas. In *Proceedings of the 2019 ACM SIGIR*

International Conference on Theory of Information Retrieval, pages 11–18.

Hunter McNichols, Mengxue Zhang, and Andrew Lan. 2023. Algebra error classification with large language models. *arXiv preprint arXiv:2305.06163*.

Jordan Meadows and André Freitas. 2021. Similarity-based equational inference in physics. *Physical Review Research*, 3(4):L042010.

Jordan Meadows and Andre Freitas. 2022. A survey in mathematical language processing. *arXiv preprint arXiv:2205.15231*.

Jordan Meadows, Zili Zhou, and Andre Freitas. 2022. Physnlu: A language resource for evaluating natural language understanding and explanation coherence in physics. *arXiv preprint arXiv:2201.04275*.

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.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2020. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial nli: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4885–4901.

Vít Novotnỳ and Michal Štefánik. 2022. Combining sparse and dense information retrieval. *Proceedings of the Working Notes of CLEF*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Judea Pearl. 2009. Causal inference in statistics: An overview. *Statistics surveys*, 3:96–146.

Judea Pearl. 2022. Direct and indirect effects. In *Probabilistic and causal inference: The works of Judea Pearl*, pages 373–392.

Shuai Peng, Ke Yuan, Liangcai Gao, and Zhi Tang. 2021. Mathbert: A pre-trained model for mathematical formula understanding. *arXiv preprint arXiv:2105.00377*.

Markus N Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. 2020. Mathematical reasoning via self-supervised skip-tree training. *arXiv preprint arXiv:2006.04757*.

Abhilasha Ravichander, Yonatan Belinkov, and Eduard Hovy. 2021. Probing the probing paradigm: Does probing accuracy entail task relevance? In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3363–3377.

Anja Reusch, Maik Thiele, and Wolfgang Lehner. 2022. Transformer-encoder and decoder models for questions on math. *Proceedings of the Working Notes of CLEF 2022*, pages 5–8.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online. Association for Computational Linguistics.

Julia Rozanova, Marco Valentino, Lucas Cordeiro, and André Freitas. 2023a. Interventional probing in high dimensions: An nli case study. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 2444–2455.

Julia Rozanova, Marco Valentino, and Andre Freitas. 2023b. Estimating the causal effects of natural logic features in neural nli models.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*.

Viktor Schlegel, Goran Nenadic, and Riza Batista-Navarro. 2023. A survey of methods for revealing and overcoming weaknesses of data-driven natural language understanding. *Natural Language Engineering*, 29(1):1–31.

Goyal Shreya, Sumanth Doddapaneni, Mitesh M Khapra, and Balaraman Ravindran. 2022. A survey of adversarial defences and robustness in nlp. *ACM Computing Surveys*.

Alessandro Stolfo, Zhijing Jin, Kumar Shridhar, Bernhard Schölkopf, and Mrinmaya Sachan. 2022. A causal framework to quantify the robustness of mathematical reasoning with language models. *arXiv preprint arXiv:2210.12023*.

Damien Teney, Ehsan Abbasnejad, Kushal Kafle, Robik Shrestha, Christopher Kanan, and Anton Van Den Hengel. 2020. On the value of out-of-distribution testing: An example of goodhart's law. *Advances in Neural Information Processing Systems*, 33:407–417.

Thi Hong Hanh Tran, Matej Martinc, Antoine Doucet, and Senja Pollak. 2022. Ijs at textgraphs-16 natural language premise selection task: Will contextual information improve natural language premise selection? In *Proceedings of TextGraphs-16: Graph-based Methods for Natural Language Processing*, pages 114–118.

Marco Valentino, Deborah Ferreira, Mokanarangan Thayaparan, André Freitas, and Dmitry Ustalov. 2022a. 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.

Marco Valentino, Mokanarangan Thayaparan, Deborah Ferreira, and André Freitas. 2022b. Hybrid autoregressive inference for scalable multi-hop explanation regeneration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11403–11411.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Victor Veitch, Dhanya Sridhar, and David Blei. 2020. Adapting text embeddings for causal inference. In *Conference on Uncertainty in Artificial Intelligence*, pages 919–928. PMLR.

Pablo Villalobos, Jaime Sevilla, Lennart Heim, Tamay Besiroglu, Marius Hobbhahn, and Anson Ho. 2022. Will we run out of data? an analysis of the limits of scaling datasets in machine learning. *arXiv preprint arXiv:2211.04325*.

Sean Welleck, Jiacheng Liu, Jesse Michael Han, and Yejin Choi. 2021. Towards grounded natural language proof generation. In *MathAI4Ed Workshop at NeurIPS*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Liuyi Yao, Zhixuan Chu, Sheng Li, Yaliang Li, Jing Gao, and Aidong Zhang. 2021. A survey on causal inference. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(5):1–46.

Chunchun Zhao and Sartaj Sahni. 2019. String correction using the damerau-levenshtein distance. *BMC bioinformatics*, 20(11):1–28.

Wei Zhong, Jheng-Hong Yang, and Jimmy Lin. 2022. Evaluating token-level and passage-level dense retrieval models for math information retrieval. *arXiv preprint arXiv:2203.11163*.

A Training

Transformer encoders with a binary sequence classification layer are fine-tuned for 12 epochs on a 16GB Tesla V100, with a batch size of 8, and a learning rate of 5e-7, via the Transformers library (Wolf et al., 2019). We use adapted versions of the public[2] training scripts. Tokenizers pad up to a max length of 256, and the best model by F1 is selected after training. We train 25 models stemming from 5 encoders: BERT-base-uncased (Devlin et al., 2018), BERT-base-cased, SciBERT cased and uncased (Beltagy et al., 2019), and MathBERT (Peng et al., 2021). SciBERT is a version of BERT pretrained on scientific text. MathBERT is initialised on BERT-base-uncased, and pretrained on three masked language modelling tasks related to the structure of equation operator trees (Mansouri et al., 2019), and the relationship between equations and their natural language context. It delivers state-of-the-art results in formula search (Zhong et al., 2022).

B Algorithm for Premise Generation

Algorithm 1 is implemented to generate premises with computer algebra. Functions specific to the computer algebra system such as *e.g.,* Symbol and Function, are used directly in the algorithm, and indirectly to compose functions such as differentiate, integrate, etc.

C Algorithm for Derivation Generation

Algorithm 2 relies on Algorithm 1, in order to derive subsequent equations. It relies on two other procedures other than Step. The EquationDistribution function relies on the hyperparameter p_h, which controls the frequency that recent equations are sampled as a cubic function of p_h. The ExtractDerivation function is responsible for collecting all related steps from the initial longer derivation, such that a final self-contained derivation is obtained. This derivation must match the desired length, L_f.

Hyperparameters. We rely on other hyperparameters to control **1.** the selection bias towards operations being applied to more recent equations, **2.** the bias towards operators of a particular arity, and **3.** bias towards other operator subcategories. Considering **1.**, in the 2-arity two annotation format ['operator', operand 1, operand 2], operand 1 is always an equation index. This is also true for 1-arity, and 0-arity does not require an operand. An equation is randomly sampled from a non-repeating set of derived equations. The *history hyperparameter*,

[2]https://huggingface.co/docs/Transformers/tasks/sequence_classification

Algorithm 1 Generate Premise Equation

Requires a global vocabulary of letters, $\mathcal{V}$, and operations *e.g.,* cos, sin, etc.

```
1: procedure PREMISE($\mathcal{M}_c$)
2:     $\mathcal{S} \leftarrow$ [Symbol($v$) for $v$ in $\mathcal{V}$]
3:     $\mathcal{R}_1 \leftarrow$ [Cos, Sin, Exp, Log]
4:     $\mathcal{R}_2 \leftarrow$ [Add, Minus, Times, Power, Divide, Differentiate, Integrate]
5:     arity $\leftarrow$ random.choice([1,2])
6:     if arity $= 1$ then
7:         $R \leftarrow$ random.choice($\mathcal{R}_1$)
8:         $S \leftarrow$ random.choice($\mathcal{S}$)
9:         RHS $\leftarrow R(S)$
10:        LHS $\leftarrow$ random.choice([$s$ for $s$ in $\mathcal{S}$ if $s \neq S$])
11:    else if arity $= 2$ then
12:        $R \leftarrow$ random.choice([$r$ for $r$ in $\mathcal{R}_2$ if $r$ not in [Differentiate, Integrate]])
13:        $S_1 \leftarrow$ random.choice($\mathcal{S}$)
14:        $S_2 \leftarrow$ random.choice([$s$ for $s$ in $\mathcal{S}$ if $s \neq S_1$])
15:        RHS $\leftarrow R(S_1, S_2)$
16:        LHS $\leftarrow$ random.choice([$s$ for $s$ in $\mathcal{S}$ if $s$ not in [$S_1$, $S_2$]])
17:    end if
18:    complexity $\leftarrow$ random.choice(range($\mathcal{M}_c$))
19:    for $i \in$ range(complexity) do
20:        arity $\leftarrow$ random.choice([1,2])
21:        if arity $= 1$ then
22:            $R \leftarrow$ random.choice($\mathcal{R}_1$)
23:            RHS $\leftarrow R($RHS$)$
24:        else if arity $= 2$ then
25:            $R \leftarrow$ random.choice($\mathcal{R}_2$)
26:            $S \leftarrow$ random.choice($\mathcal{S}$)
27:            RHS $\leftarrow R($RHS$, S)$
28:        end if
29:    end for
30:    LHS $\leftarrow$ Function(LHS)(*tuple(RHS.free_symbols))
31:    premise $\leftarrow$ Eq(LHS, RHS)
32:    return premise
33: end procedure
```

p_h, clones an equation in the list, through a cubic function of its step-wise chronological position, as described above. With our default settings, the last equation in a list of three is twice as likely to be selected as input than the first. This emulates mathematicians working with recent equations, but having to occasionally sample from distant results. Other hyperparameters work similarly, by repeating elements of lists. Considering **2.**, we bias towards 2-arity, as those contain calculus, and considering **3.** we bias towards substitution operations, differentiation, and integration.

Collecting scattered derivations. A sequence of operations applied to premises can result in unlinked derivations. These can merge to form larger derivations by substituting expressions from one into another. If a merging operation does not occur, the derivations remain separated, and each may be treated independently. Briefly, we traverse a single generated chain reverse-chronologically, using the equation numbers in annotations as guides for determining dependency. This is handled by the ExtractDerivation function.

Algorithm 2 Generate Equational Reasoning

```
1: procedure STEP($\mathcal{D}, p_0, p_1, p_2, p_h, p_r, p_e, p_c, p_s$)
2:     $D \leftarrow$ [$i$[0] for $i$ in $\mathcal{D}$]
3:     $A \leftarrow$ [$i$[1] for $i$ in $\mathcal{D}$]
4:     $\mathcal{R}_0 \leftarrow$ [Premise] + [RenamingPremise]$\times p_r$
5:     $\mathcal{R}_1 \leftarrow$ [Cos, Sin, Exp, Log, Expand] + [EvaluateDerivatives, EvaluateIntegrals]$\times p_e$
6:     $\mathcal{R}_2 \leftarrow$ [Add, Minus, Times, Divide, Power] + [Differentiate, Integrate]$\times p_c$
           + [SubsLHSForRHS, SubsRHSForLHS]$\times p_s$
7:     elements $\leftarrow$ numbers, variables, and subexpressions from $D$
8:     arity $\leftarrow$ random.choice([0]$\times p_0$ + [1]$\times p_1$ + [2]$\times p_2$)
9:     if arity $= 0$ then
10:        $R \leftarrow$ random.choice($\mathcal{R}_0$)
11:        equation $\leftarrow R$
12:        annotation $\leftarrow R$.__name__
13:    else if arity $= 1$ then
14:        $R \leftarrow$ random.choice($\mathcal{R}_1$)
15:        $e_1 \leftarrow$ random.choice(EquationDistribution($D, p_h$))
16:        equation $\leftarrow R(e_1)$
17:        $n \leftarrow D$.index($e_1$)
18:        annotation $\leftarrow$ [$R$.__name__, $n+1$]
19:    else if arity $= 2$ then
20:        $R \leftarrow$ random.choice($\mathcal{R}_2$)                ▷ $R$ depends on the length of $D$
21:        $e_1 \leftarrow$ random.choice(EquationDistribution($D, p_h$))
22:        $e_2 \leftarrow$ random.choice(elements)                     ▷ $e_2$ will vary depending on $R$
23:        equation $\leftarrow R(e_1, e_2)$
24:        $n \leftarrow D$.index($e_1$)
25:        annotation $\leftarrow$ [$R$.__name__, $n+1$, $e_2$]
26:    end if
27:    if equation is valid then                                   ▷ validity depends on various checks
28:        return equation
29:    else
30:        return None
31:    end if
32: end procedure
33: while True do
34:    $\mathcal{D} \leftarrow$ [(Premise($\mathcal{M}_c$), "premise")]        ▷ generate first step using Algorithm 1
35:    while len($\mathcal{D}$) $< L_i$ do                                ▷ $L_i$ is an initial length of the derivation
36:        step $\leftarrow$ Step($\mathcal{D}, p_0, p_1, p_2, p_h, p_r, p_e, p_c, p_s$)
37:        if step is not None then
38:            $\mathcal{D}$.append(step)
39:        end if
40:    end while
41:    derivation $\leftarrow$ ExtractDerivation($\mathcal{D}$)
42:    if len(derivation) $= L_f$ then                  ▷ $L_f \leq L_i$ is the desired length of the derivation
43:        break
44:    end if
45: end while
46: $\mathcal{D} = derivation$
```