A Symbolic Framework for Evaluating Mathematical Reasoning and Generalisation with Transformers

Anonymous ACL submission

Abstract

This paper proposes a methodology for generating and perturbing detailed derivations of equations at scale, aided by a symbolic engine, to evaluate the generalisability of Transformers to out-of-distribution mathematical reasoning problems. Instantiating the framework in the context of sequence classification tasks, we compare the capabilities of GPT-4, GPT-3.5, and a canon of fine-tuned BERT models, exploring the relationship between specific operators and generalisation failure via the perturbation of reasoning aspects such as symmetry and variable surface forms. Surprisingly, our empirical evaluation reveals that the average in-distribution performance of fine-tuned models surpasses GPT-3.5, and rivals GPT-4. However, perturbations to input reasoning can reduce their performance by up to 80 F1 points. Overall, the results suggest that the in-distribution performance of smaller open-source models may potentially rival GPT by incorporating appropriately structured derivation dependencies during training, and highlight a shared weakness between BERT and GPT involving a relative inability to decode indirect references to mathematical entities. We release the full codebase, constructed datasets, and fine-tuned models to encourage future progress in the field[1].

Out-of-distribution generalisation in Transformers (Vaswani et al., 2017) is a fundamental and desirable property (Schlegel et al., 2023; Belinkov, 2022; Teney et al., 2020), especially in domains that require rigorous and controlled reasoning such as mathematics, physics, biomedicine, and software verification (Frieder et al., 2023; Lee et al., 2022; Valentino et al., 2022b; Lewkowycz et al., 2022; Drori et al., 2022; Welleck et al., 2021; Kumar et al., 2020). Various strategies have been proposed to evaluate model generalisability, including direct input manipulation (Rozanova et al., 2023b; Stolfo et al., 2022; Nie et al., 2020; Kaushik et al., 2019; Welleck et al., 2022) and probing on the internal representation (Rozanova et al., 2023a; Ravichander et al., 2021; Elazar et al., 2021; Veitch et al., 2020). However, the adoption of such methods for evaluating generalisation on complex, multi-step reasoning problems is still limited. 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). Particularly in the mathematical domain, these hurdles impact the scope and reliability of causality and robustness studies (Pearl, 2009; Shreya et al., 2022).

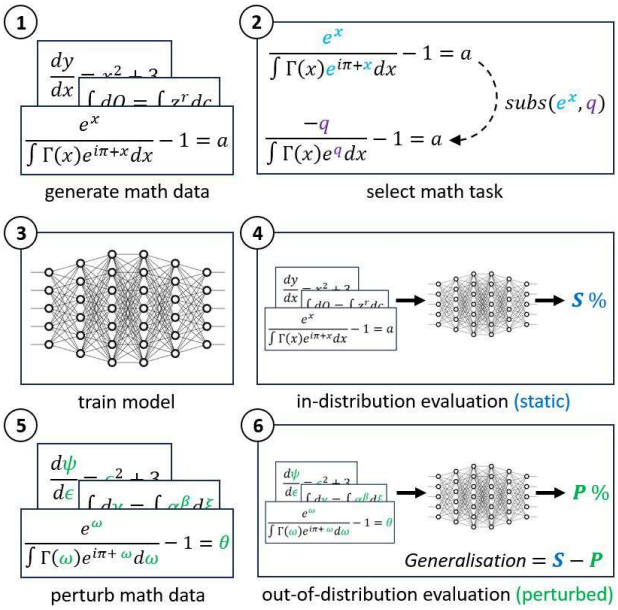

Figure 1: We present a framework for generating and perturbing high-quality mathematical derivations at scale to systematically evaluate mathematical reasoning and generalisation in Transformers.

We leverage the rich environment of symbolic engines to design a data generation and evaluation

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

framework that can produce high-quality mathematical reasoning steps possessing diverse symbolic properties at scale. In particular, strict symbolic rules offer a systematic approach to *perturbing* mathematical reasoning, and hence evaluating the generalisation of neural models to out-of-distribution textual aspects such as symmetry and variable surface forms. This allows the exploration of deep relationships between semantic and syntactic elements of math reasoning and model generalisability across diverse subdomains, extending beyond the limited interventional scope of previous works (Stolfo et al., 2022; Welleck et al., 2022; Patel et al., 2021; Ribeiro et al., 2020; Kaushik et al., 2019; Yao et al., 2021). Additionally, we dialogue with 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 (Meurer et al., 2017; Wolfram, 1999) facilitate the generation of annotated mathematical reasoning, which allows the construction of high-quality datasets for various tasks. We combine 18 symbolic operators with rules that guide the exploration of equational state spaces and generate derivations, then perturb and adapt them for exemplar entailment tasks. In this case, these are sequence classification tasks that focus on operator usage in reasoning chains.

To demonstrate our approach, we fine-tune a canon of BERT-based models used in mathematical language processing (Li et al., 2023; McNichols et al., 2023; Meadows and Freitas, 2023; Zhong et al., 2022), and test in-context learning methods with GPT-3.5 and GPT-4, to determine their capacity for recognising coherent math reasoning (within this scope), and to abstract fundamental properties impacting their ability to generalise. To summarise, the paper offers the following contributions:

(1.) A general approach to generating annotated derivations of controllable complexity levels, involving premise equation generation (Alg. 2) and the sequential application of operators to prior equations to derive new results (Alg. 3).

(2.) A systematic and scalable methodology to perturb various aspects of mathematical data including syntax and semantics, implementing several perturbations for evaluation.

(3.) An experimental framework for evaluating the out-of-distribution generalisation of models on mathematical reasoning tasks (Fig. 1).

(4.) Example instantiation of the framework involving sequence classification tasks. The generated datasets include static and perturbed derivations totalling over 200K examples.

(5.) An extensive evaluation of various BERT-based and GPT models culminating in a discussion relating the limited generalisability of models with respect to key operators and mathematical content.

In short, the proposed mathematical data generation and perturbation approach may be integrated into evaluation frameworks for the purpose of testing model generalisability to specific distribution shifts, such as specific surface forms of equations or operator usage. We apply the framework to demonstrate the brittleness of fine-tuned encoder models, and reveal underlying weaknesses shared by both BERT and GPT.

1 Related Work

Computer algebra. SymPy (Meurer et al., 2017) is a computer algebra system incorporated within numerous approaches. For example, Chen et al. (2022) solve numerical reasoning tasks including simple math elements such as numbers, by 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 without exploring generalisability aspects. 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. Lample and Charton (2019) train a model to integrate and solve differential equations, but do not explore generalisation (Davis, 2019). Welleck et al. (2022) conduct similar experiments using a single model and a single operator (integration) on a single task. We consider 18 operations, 7 models, multiple tasks, and emphasize multi-step equational reasoning.

Reasoning with mathematical language. Transformers (Saxton et al., 2019; Clark et al., 2020; Rabe et al., 2020) defined the state-of-the-art in multiple subdomains and tasks in mathematical language processing (Azerbayev et al., 2023; Meadows and Freitas, 2023; Lewkowycz et al., 2022; Drori et al., 2022). Transformer encoder models obtain leading performance in related tasks (Ferreira et al., 2022; Lai et al., 2022; Zhong et al., 2022; Peng et al., 2021; Valentino et al., 2022a;

Tran et al., 2022; Reusch et al., 2022; Novotnỳ and Štefánik, 2022). The evaluation of the mathematical capabilities of GPT models, and the comparison between GPT and smaller fine-tuned models when deriving equations, has been considered elsewhere (Meadows et al., 2023; Frieder et al., 2023; Azerbayev et al., 2023).

Data augmentation, synthetic benchmarks, and evaluation frameworks. Numerous approaches exist related to evaluating the mathematical and symbolic capabilities and robustness of models (Li et al., 2020). 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 in causal analysis (Pearl, 2022; Christiansen et al., 2021; Patel et al., 2021; Ribeiro et al., 2020). Mirroring other notable work (Welleck et al., 2022), their approach focuses on one or two task-dependent perturbations. Our approach to generating and perturbing data is largely task-independent, and allows for the complex augmentation of operators, variables, expressions, and equations in multi-hop reasoning chains. INT (Wu et al., 2020) is a similar generation metric more closely aligned with formal theorem proving (Polu and Sutskever, 2020; Moura et al., 2015). Our work is aligned with computer algebra systems (Meurer et al., 2017) and more applied mathematical domains (*e.g.,* physics, engineering), and includes calculus.

2 Generating and Perturbing Derivations with Symbolic Engines

The data generation approach outputs and perturbs multi-step reasoning involving step annotations and equations. A model learns mathematical reasoning in the context of a given task, is evaluated on an in-distribution test set, then each element of that set is symbolically perturbed and the difference in model inference due to the perturbation contributes to a generalisability measure for the model (Fig. 1).

To outline this process, given a vocabulary of symbols $\mathcal{V}$ and set of computer algebra operations $\mathcal{R}$, each are sampled from to ultimately generate an ordered list of steps $s_i \in \mathcal{D}$, where $\mathcal{D}$ represents the output derivation. An initial reasoning step $s_1 = (\text{premise}, \text{annotation})$ is generated such that $\mathcal{D} = [s_1]$. An operation $r \in \mathcal{R}$ is sampled, which in its most general form accepts two operands (arity 2). The first operand is an equation $s_{j,1}$ from tuple $s_j \in \mathcal{D}$. A suitable secondary variable ($\in \mathcal{V}$), expression, or equation operand m is extracted from $\mathcal{D}$, and the next equation is generated by applying operation r through $s_{i+1,1} = r(s_{j,1}, m)$. The annotation $s_{i+1,2}$ is also a list containing (most generally) the name of the operation, the equation index, and secondary operand, such that $s_{i+1,2} = [r, j, m']$ (where m' is a variable/expression string or equation index representing operand m). Therefore, step $s_{i+1} = (r(s_{j,1}, m), [r, j, m])$. If $\mathcal{D} = [s_1]$, then $i = j = 1$, and the derivation updates such that $\mathcal{D} = [s_1, s_2]$. This process repeats until the derivation reaches a target length.

Given a specific task (*e.g.,* sequence classification), a derivation $\mathcal{D}$ is adapted to form model input $\mathcal{D}'$, such that a static test set is sampled from the same distribution as the training set or in-context examples. This static set X contains task examples $\mathcal{D}'_i$ and labels. For all $\mathcal{D}'_i \in X$, a perturbation $\mathcal{P}$ is applied to corresponding initial derivation $\mathcal{D}_i$ to form a perturbed derivation $\mathcal{P}(\mathcal{D}_i)$, which is similarly adapted to form an out-of-distribution perturbed task example $\mathcal{P}(\mathcal{D}_i)' \in P$, where P is the out-of-distribution test set corresponding to static set X, such that $\mathcal{P} : X \rightarrow P$.

Now that we have static (in-distribution) and perturbed (out-of-distribution) reasoning pairs (X_i, P_i), for a given model $\mathcal{M}$ we can evaluate its generalisability by comparing its static and perturbed predictions, respectively $\mathcal{M}(X_i)$ and $\mathcal{M}(P_i)$. Together with the respective labels, these outputs allow comparisons between aggregate scores (Tab. 1 and 2), but also a pair-wise analysis involving more sophisticated logic involving $\mathcal{M}(X_i)$ and $\mathcal{M}(P_i)$, which may include perturbations other than $\mathcal{P}$ (Tab. 3).

This provides a highly controllable symbolic framework for evaluating mathematical generalisation capabilities of models. The approach produces mathematical reasoning at scale, and can both improve the depth of mathematical corpora through the generation of underrepresented reasoning, and serve as the backbone for testing model generalisability in numerous settings.

2.1 Premise Generation

To generate premise equations we use a vocabulary (uppercase and lowercase English characters, excluding {i, e, d, O}) and a set of 18 operators defined within the symbolic engine, separated by arity. For instance, arity "0" represents the introduction of a premise. Arity 1 includes operations

that only accept a single variable, expression, or equation (*e.g.,* simplification). Arity 2 includes those such as addition and integration.

Alg. 2 (Appendix D) describes how operators are recursively applied to vocabulary elements and expressions to produce premises such as: $z(n, f) = f + n$, $J(p, w) = e^{p^w}$, and $Q(x) = \log(x)$. To give a brief description, a symbol is first sampled from the vocabulary. Then an operator with a specific arity is selected, which is applied to the symbol (potentially after selecting another symbol depending on the arity) to form the RHS of an equation. If $\mathcal{C} = 1$, this current RHS will feature as the final RHS of the premise. Otherwise, operators will be re-selected and re-applied to the current RHS up to $\mathcal{C} - 1$ times. Once the premise has reached a sufficient complexity, a unique symbol is sampled for the equation LHS, which represents a function of the RHS variables, and the LHS and RHS are conjoined as a SymPy equation.

2.2 Derivation Generation

Algorithm 1 Generate Derivation Step

procedure STEP($\mathcal{D}$)
  Initialise operator sets $\mathcal{R}_0$, $\mathcal{R}_1$, and $\mathcal{R}_2$, and set sampling probabilities
  Sample arity $a \in \{0, 1, 2\}$
  if $a = 0$ **then**
    Sample operator $R \in \mathcal{R}_0$
    Generate equation using R, annotation
  else if $a = 1$ **then**
    Sample unary operator $R \in \mathcal{R}_1$
    Sample equation e from $\mathcal{D}$
    Generate equation $R(e)$ and annotation
  else if $a = 2$ **then**
    Sample binary operator $R \in \mathcal{R}_2$
    Sample equation e from $\mathcal{D}$
    Sample variable, expression, or equation, m, from $\mathcal{D}$
    Generate equation $R(e, m)$ and annotation
  end if
  return (equation, annotation) **if** equation is valid **else** None
end procedure

To generate a derivation, a premise equation generated by Alg. 2 and an annotation are initially stored as a tuple (equation, annotation) within a list $\mathcal{D}$. This list is input to the `Step` procedure (Alg. 1) which considers any mathematical elements defined in $\mathcal{D}$ so far, and uses them as input to the operators to generate new (equation, annotation) tuples. Tuples containing equations that pass validity checks such as maximum character length (250), LHS/RHS redundancies (*e.g.,* $x = x$), and checks related to integration (etc.), are appended to $\mathcal{D}$. A final derivation is outputted when $\mathcal{D}$ exceeds a target length. This process is described further in Alg. 3, including a description of its output rate (7 seconds/step on mid-range CPU), hyperparameters, and equation sampling, in Appendix E.

2.3 Perturbations

To perturb LaTeX sequences, the examples in the static set are *re-interpreted* by the computer algebra system using SymPy's `srepr` tree representation. In this paper, we consider four different perturbations for evaluation (Fig. 2). However, the compatibility with the computer algebra system allows a wide variety of perturbations that range from small-scale interventions to single variables, through to long-range interventions that target complex semantic relationships between any number of distant sequence elements. For instance, one may choose to only perturb reasoning chains that involve a premise renaming operation followed directly by integration, or square a variable and propagate that change through the entire reasoning chain. The perturbations adopted in our evaluation are as follows:

Variable Renaming (VR). For each example in the static set, we uniquely map each symbol to an out-of-vocabulary symbol sampled from 10 Greek letters (*e.g.,* $E(n, x) = n + x$ becomes $\alpha(\beta, \gamma) = \beta + \gamma$).

Expression Exchange (EE). For each example in the static set, we swap expressions either side of the equality (*e.g.,* $E(n, x) = n + x$ becomes $n + x = E(n, x)$). This reverses the overrepresentation of LHS functions in the static set.

Annotation Replacement (AR). Each example in the static set contains a correct and incorrect final equation. For each example, the operator and operands (and hence the annotation) responsible for generating the negative equation are calculated, replacing the corresponding annotation in the sequence and swapping the label (i.e. from positive to negative and vice-versa).

Equation Conversion (EC). If a sequence consists of a chain such as $\log(x)$ [SEP] x [SEP] $\frac{1}{x}$, and the implicit operation is differentiation, a random symbol is sampled from the vocabulary

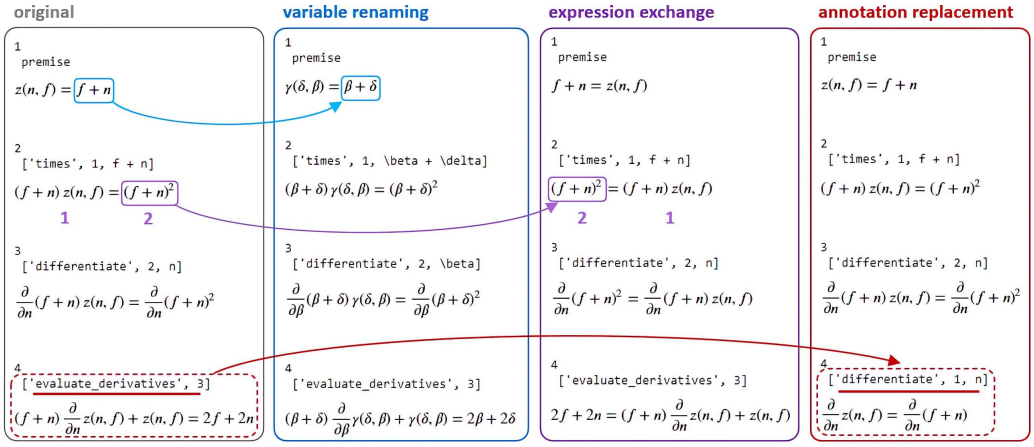

Figure 2: Example perturbations applied to a generated derivation using computer algebra.

(*e.g.,* Q), and the sequence becomes $Q(x) = \log(x)$ [SEP] x [SEP] $\frac{dQ(x)}{dx} = \frac{1}{x}$. If integrating, then the (negative) sequence becomes $\frac{dQ(x)}{dx} = \log(x)$ [SEP] x [SEP] $Q(x) = \frac{1}{x}$.

3 Sequence Classification Tasks

The data generation approach (Alg. 1-3) outputs math reasoning which may then be adapted for specific tasks. As such, we instantiate the general framework described in Fig. 1 in the context of *two sequence classification tasks*. Task examples, dataset sizes, and model input (Fig. 4) are described in Appendix B.

Derivation Step Classification. Given multiple steps of a derivation, such as those in Fig. 2 and 4, the final equation has a 50% chance of being replaced with a similar equation that is *associated with a different annotation*, and the model must classify whether the overall derivation is coherent (*i.e.,* whether the annotations match the equations and overall reasoning). Incoherent equations associated with a negative label are generated by applying a different operator, or the same operator with different inputs, to a previous derivation equation through `Step` (Alg. 1). To solve this task a model must learn the necessary *inter-statement dependencies* required to form the final equation in the derivation, guided by the final annotation. These dependencies are a crucial aspect of derivations and equational reasoning.

Calculus Classification. Given a simplified sequence containing only a *premise expression*, a *variable*, and a *final expression* (Fig. 4), where the final expression is the premise differentiated or integrated with respect to the variable (*e.g.,* premise: $\log(x)$, var: x, final: $1/x$), the final expression has a 50% chance of being swapped with a similar but incorrect expression. The negative examples are generated by differentiating/integrating the dataset premises by fixing the variable and changing the premise, or vice versa. To select the expression for the final expression swap to form the incoherent sequence (negative label), these expressions are ranked by their Damerau-Levenshtein distance (Zhao and Sahni, 2019; Meadows and Freitas, 2021) from the ground truth. For example, the expression $-T + sin(U)$ is differentiated with respect to T to give -1. The expression corresponding to the negative label is 1.

4 Evaluation

In this section, we discuss how the scores obtained by the BERT and GPT models reflect their reasoning capabilities within the scope of the classification tasks described in the previous section.

Training and prompts. Details related to fine-tuning BERT and prompting GPT are given in Appendix A. To summarise, for a single pre-trained BERT model, *five* fine-tuned models are trained for each of the task variations. For instance, in *Derivation Step Classification*, a model is trained *per number of steps* in the input derivations (*i.e.,* 2, 3, and 4 steps). For *Calculus Classification* a model is trained *per operation* (*i.e.,* differentiation and integration). The GPT models are given 4-shot prompts. While we acknowledge that chain-of-thought (Wei et al., 2022) and tree-of-thought (Yao et al., 2023) methods may lead to improved mathematical inference, we instead rely on a simple few-shot prompt (Tab. 4) and focus on the evaluation framework and data generation pipeline.

Further generalisation. In addition to the out-

	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**)	87.7	88.9	87.0	88.1	87.0	88.0	87.5	88.7	-	-	-	-
BERT-base-uncased (**s=3**)	78.9	78.7	71.9	71.0	69.1	66.0	53.7	50.6	68.4	69.0	-	-
BERT-base-uncased (**s=4**)	58.8	63.6	55.0	60.3	56.4	60.3	42.4	48.1	65.7	62.2	52.8	29.8
BERT-base-cased (**s=2**)	87.2	88.5	81.9	83.2	85.3	86.1	85.5	87.2	-	-	-	-
BERT-base-cased (**s=3**)	78.2	77.3	68.8	64.5	65.0	58.9	54.5	49.6	54.6	30.5	-	-
BERT-base-cased (**s=4**)	66.8	71.7	58.5	61.5	62.6	***67.2***	43.3	53.1	71.9	73.9	54.3	21.8
MathBERT (**s=2**)	83.2	82.0	76.2	70.6	79.0	75.7	78.5	76.0	-	-	-	-
MathBERT (**s=3**)	84.2	83.9	69.1	64.5	63.3	52.2	66.3	64.0	67.4	58.7	-	-
MathBERT (**s=4**)	67.1	68.4	59.5	52.6	62.3	62.1	48.5	47.9	68.6	68.0	51.8	29.0
SciBERT-uncased (**s=2**)	92.5	92.6	72.9	70.4	86.8	86.1	90.0	90.2	-	-	-	-
SciBERT-uncased (**s=3**)	88.9	***89.4***	82.1	***81.9***	70.3	66.4	70.9	***72.2***	80.6	***81.8***	-	-
SciBERT-uncased (**s=4**)	76.3	***76.5***	69.5	66.8	68.6	65.9	60.7	***59.6***	76.9	***77.9***	59.3	57.4
SciBERT-cased (**s=2**)	92.6	**93.1**	85.3	87.1	89.8	**90.2**	91.0	**91.7**	-	-	-	-
SciBERT-cased (**s=3**)	77.2	72.4	72.7	67.2	61.0	44.1	50.8	29.5	52.9	12.8	-	-
SciBERT-cased (**s=4**)	71.0	70.9	65.1	64.6	66.6	65.4	47.0	42.9	77.9	74.9	52.7	11.0
Encoder Average (**s=2**)	88.6	89.0	80.7	79.9	85.6	85.3	86.5	86.8	-	-	-	-
Encoder Average (**s=3**)	81.5	80.3	72.9	69.8	65.7	57.5	59.2	53.2	-	-	-	-
Encoder Average (**s=4**)	68.0	70.2	61.5	61.2	63.3	64.2	48.4	50.3	-	-	-	-
GPT-3.5 (**s=2**)	66.0	72.6	65.5	72.5	59.0	65.3	53.0	63.3	-	-	-	-
GPT-3.5 (**s=3**)	57.0	64.2	61.5	67.0	60.5	65.5	46.0	54.2	56.5	64.5	-	-
GPT-3.5 (**s=4**)	51.5	59.1	49.5	56.3	54.0	59.6	44.5	52.8	56.0	62.7	59.0	67.7
GPT-4 (**s=2**)	88.0	88.5	87.5	**88.2**	82.5	81.1	64.5	66.4	-	-	-	-
GPT-4 (**s=3**)	77.5	77.4	77.5	76.7	78.5	***77.2***	50.0	55.0	73.5	77.4	-	-
GPT-4 (**s=4**)	68.0	68.0	69.0	***69.6***	66.0	64.6	42.0	42.6	76.0	76.9	77.5	***80.2***
Encoder (steps avg)	79.4	79.8	71.7	70.3	71.5	69.0	64.7	63.4	-	-	-	-
GPT-3.5 (steps avg)	58.2	65.3	58.8	65.3	57.8	63.5	47.8	56.8	-	-	-	-
GPT-4 (steps avg)	77.8	78.0	78.0	78.2	75.7	74.3	52.2	54.7	-	-	-	-

Table 1: *Derivation Step Classification* task results. 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.

of-distribution perturbations applied to input sequences for each task, a model that can sufficiently generalise the underlying reasoning should be able to solve (on average) mathematically less complex versions of problems encountered during training. In *Derivation Step Classification*, we evaluate models trained on derivations with a fixed step count on a set of derivations composed of a lower number of steps. This is represented in the **s - 1** and **s - 2** columns in Tab. 1 given initial step count, **s**. In *Calculus Classification*, where models are exposed to examples comprising *at least two* variables, (*e.g.,* $\cos(ax) - z$) we generate a set of easier problems with 1.5k examples consisting of only one variable (*e.g.,* $\cos(x)$), corresponding to the **Easy** column of Tab. 2. These out-of-distribution datasets complement the perturbations in the following discussion.

GPT-4 rivals in-distribution performance of fine-tuned BERT-based models while demonstrating better generalisation. Assuming a suitably descriptive few-shot prompt, where necessary context is provided through either the task description or in-context examples (Appendix A), GPT-4 can rival the average static scores of the fine-tuned encoder models, and surpass them on out-of-distribution test sets, *even without chain-of-thought prompting*. This is shown by the *Derivation Step Classification* results (Tab. 1). For instance, SciBERT-cased (**s=4**) scores 11% F1 when classifying sequences with **s=2** steps. GPT-4 obtains 80% F1 in this case. Similar generalisation is observed on the **VR** (*Variable Renaming*) pertur-

	Static		VR		EC		Easy	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1
BERT-base-uncased (**int**)	90.0	90.7	68.8	70.4	75.1	78.0	62.7	***72.9***
BERT-base-uncased (**diff**)	75.9	80.3	64.9	73.3	62.2	***69.8***	55.1	69.1
BERT-base-cased (**int**)	93.0	93.4	71.6	**77.7**	85.2	**86.7**	63.8	71.8
BERT-base-cased (**diff**)	74.2	77.9	64.2	72.4	60.3	64.9	56.7	***69.6***
MathBERT (**int**)	92.2	92.3	74.4	75.8	74.4	71.8	58.6	68.6
MathBERT (**diff**)	84.7	85.9	59.7	48.1	58.4	47.3	56.1	50.0
SciBERT-uncased (**int**)	96.8	96.8	65.6	74.4	54.1	15.8	62.6	71.1
SciBERT-uncased (**diff**)	91.8	92.3	72.6	***76.5***	66.8	58.1	55.2	67.8
SciBERT-cased (**int**)	97.1	**97.2**	68.1	75.8	54.2	17.0	58.0	67.1
SciBERT-cased (**diff**)	92.3	***92.7***	70.9	76.5	65.4	54.6	61.5	72.3
Encoder Average (**int**)	93.8	93.2	69.7	74.8	68.6	53.7	61.1	70.3
Encoder Average (**diff**)	83.8	85.8	66.5	69.4	62.6	58.9	56.9	65.8
GPT-3.5 (**int**)	49.5	56.3	49.5	56.3	51.5	60.1	54.5	58.1
GPT-3.5 (**diff**)	49.0	55.3	48.5	54.2	53.0	65.7	54.5	59.2
GPT-4 (**int**)	64.0	60.0	67.0	64.1	66.5	68.5	57.5	56.4
GPT-4 (**diff**)	59.5	55.2	61.0	57.1	66.5	72.9	68.5	66.3
Encoders (int/diff avg)	88.8	89.5	68.1	72.1	65.6	56.3	59.0	68.1

Table 2: *Calculus Classification* task results. Bold numbers denote highest F1 scores for **integration** derivations. Bold italic denotes highest ***differentiation*** scores.

bation data, likely due to GPT-4's exposure to vast vocabularies of mathematical symbols (*e.g.,* Greek symbols), and the **EE** (*Expression Exchange*) set, likely due to GPT-4's exposure to equations with RHS functions which lessens the impact of LHS function bias.

GPT-4 can fail to predict mathematical coherence from in-context examples alone. The *Calculus Classification* task includes minimalistic sequences without operation annotations. Surprisingly, while GPT-4 achieves the best performance on *Derivation Step Classification*, competitive performance is not observed in *Calculus Classification* despite its lower complexity. We attribute this to the fact that, unlike BERT, GPT is not fine-tuned on a specific operation, and in-context examples alone might not contain enough information to consistently discriminate whether a particular sequence involves either differentiation or integration. This is evidenced by the fact that both GPT models score *higher* on the **EC** (*Equation Conversion*) set. The EC perturbation changes nothing about the operation being performed, but *adds context* by writing (*e.g.*) differentiated expressions as equations with a LHS that includes $\frac{d}{dx}$. F1 scores in GPT models *increase by up to 12 points* in this case, while BERT-based scores *decrease by up to 80 points* (Tab. 2). To reinforce this, in *Derivation Step Classification*, both GPT models obtain comparatively lower scores on the **AR** (*Annotation Replacement*) set. This is because sufficient context has been provided only for an operator that differs to the main annotation operator. GPT only learns the format of the sequences and the expected output for the task in this case. However, static performance is maximised by designing the prompt in this manner (Tab. 4).

GPT-3.5 cannot effectively classify mathematical reasoning. GPT-3.5 scores *15 less* F1 points than the average encoder score of 80% on the in-distribution set, and is notably outperformed by BERT-based models on most test sets (particularly SciBERT). A notable exception are those that contain less steps (Tab. 1), where performance generally *increases* comparative to static in-distribution scores. This contrasts with the significant corresponding performance drops observed in the BERT-based evaluation, indicating that GPT learns enough from in-context examples to generalise to derivations with less steps, and therefore has a deeper relative understanding of the underlying mathematics.

Encoder models fail to generalise. For *Derivation Step Classification*, 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 encoder models are sensitive to at least one perturbation. All models tested do not generalise to *less* derivation steps, reaching as low as 11% F1. In *Calculus Classification* static scores average 90% and perturbations decrease this by 17% (**VR**) and 33% (**EC**). All fine-tuned models fail to generalise to perturbations and simpler examples, *with* 97% *F1 scores repeatedly dropping below* 17%. Despite the in-distribution performance, this indicates their reliance on superficial patterns rather than the underlying rules of the operators.

4.1 Relating Operators to Model Generalisability via Pairwise Analysis

In addition to the above evaluation, because the framework offers in-distribution and out-of-distribution pairs that correspond to a single reasoning chain and its perturbation (Fig. 2), we can measure generalisability using alternative methods. These are discussed in Tab. 3, and further discussed in Appendix C, but we summarise our findings here.

Which operators are most difficult to learn? Substitution is dependency-wise the most complicated operation and is not associated with a fixed token (such as addition's "+"). It requires a deeper understanding of derivation structure due to a necessary reliance on dependency relations across equations. All models interpret substitution relatively poorly (**None** column, Tab. 3). Operator usage that is easier for models to recognise (and generalise) involves integration or differentiation (**All** column, Tab. 3), and these are associated with specific text spans such as "\int" or "\partial". Together, this indicates that all models struggle most when operators are *not associated with fixed text spans* or when they rely on *explicitly structured dependency relations*.

Which operations contribute to poor generalisability? We consider the proportion of examples where static predictions succeed while all perturbation predictions fail (column G, Tab. 3). For BERT models, *premise renaming* and integration/differentiation *evaluation* operations rank highly, yet this is not mirrored by GPT. Fig. 3 ex-

	Static (S)	**Generalisability** (G)	**None**	**All**
BERT	76.0 $\int_E\ R\ \int\ \partial\ \times$	3.3 $\int_E\ R\ +\ \partial_E\ -$	16.5 $S_L\ S_R\ +\ X^O\ \times$	60.8 $\int\ \partial\ \times\ -\ X^O$
MathBERT	79.7 $\int_E\ R\ \int\ \partial\ \partial_E$	9.0 $R\ \int_E\ X^O\ \partial_E\ \div$	13.2 $+\ S_L\ \div\ S_R\ \cos$	57.2 $\partial\ \int\ X^O\ +\ \div$
SciBERT	**87.8** $R\ \int_E\ \int\ -\ \div$	5.0 $R\ \div\ \partial_E\ +\ X^O$	**7.0** $S_L\ S_R\ +\ \cos\ \times$	62.7 $\int\ \partial\ -\ +\ \partial_E$
GPT-3.5	58.2 $\cos\ X^O\ \partial\ \int\ R$	2.3 $S_L\ \int_E\ S_R\ +\ X^O$	29.7 $-\ \int_E\ \times\ +\ \div$	45.5 $\cos\ X^O\ \int\ \partial\ \partial_E$
GPT-4	77.8 $\cos\ \partial\ \int\ X^O\ \int_E$	**1.7** $\cos\ \times\ \partial_E\ \div\ R$	12.0 $S_L\ S_R\ -\ R\ \times$	**64.7** $\cos\ \partial\ X^O\ \int\ \times$

Table 3: **Static** (S) represents model accuracy with respect to unperturbed examples. **Generalisability** (G) represents the percentage of examples where static predictions are correct and all perturbed predictions failed (lower is better). **None** represents examples where models failed predictions in all cases, and **All** represents the opposite. Symbols correspond to the top-5 most frequent (final) operators in each unperturbed sequence, where frequency is normalized with respect to operator count in the static set. R is a premise renaming operator. $\int$ and ∂ are integration and differentiation operators. $\int_E$ and ∂_E are respective evaluation operators. X^O is exponentiation, S_L and S_R are LHS and RHS substitutions, and arithmetic symbols have their usual meaning.

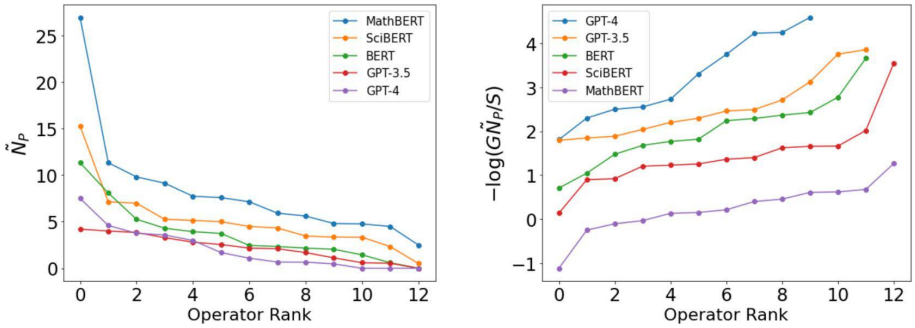

Figure 3: $\tilde{N}_P$ is the percentage of operators present in examples where models fail to generalise to perturbations. The leftmost displays how this proportion varies as a function of operator rank. The rightmost graph factors in static performance (S) and generalisability (G) scores for a clearer comparison of models.

plains this difference, displaying the proportion of operators ($\tilde{N}_P$) that contribute to examples where models generalise poorly at a given rank. For example, the highest ranking operator for MathBERT has $\tilde{N}_P > 25$. From Tab. 3 this operator performs *premise renaming*, denoted by R, and over 1/4 of examples involving R contribute to poor model generalisability. In fact for all BERT-based models, the R (and less so the int/diff evaluation) operators have a higher $\tilde{N}_P$ than others. This effect is less prominent for the GPT models. This indicates that high ranking operators have a major impact on generalisation in BERT models, but it is likely that other factors (such as the complexity of equations) are more impactful for GPT.

5 Conclusion

This research presents an approach for generating synthetic data, and a framework for evaluating the mathematical capabilities of models, which may be utilised for purposes beyond sequence classification (Valentino et al., 2023; Meadows et al., 2023). We discover that inference failures occur for BERT and GPT models when tasks require complex indirect textual references and inter-statement dependencies. This demonstrates how transformer-based models fail to appropriately infer structured information from linear text. Our findings reveal the complete generalisation failure of BERT models to simple perturbations despite their continued use in the mathematical domain, yet, the experiments reveal that BERT-related models may outperform or match GPT models in specific tasks if trained with well-structured data, despite the disparity in pre-training efforts and parameter count. Overall, this paper underscores the potential of using symbolic engines to generate extensive, high-quality mathematical datasets that may be used to explore model weaknesses, and improve mathematical reasoning and generalisation in quantitative domains.

6 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 reasoning capabilities and the biases of these models.

Chain-of-Thought. Chain-of-thought and related prompt engineering may lead to improved LLM-based mathematical reasoning. This paper focuses on the exploration and application of the evaluation framework, rather than maximising the mathematical proficiency of language models.

Derivation generation. There are irrelevant steps in some 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 the result of a premise selection operation). This limitation stems from incorrect subderivation extraction from longer chains and will be improved.

Integration. SymPy does not generate integration constants. Although we account for this within derivation generation, we currently exclude the evaluation of double (or above) integrals, and do not introduce an integration constant when generating expressions for the Calculus Classification task.

References

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

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*.

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.

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

Ernest Davis. 2019. The use of deep learning for symbolic integration: A review of (lample and charton, 2019). *arXiv preprint arXiv:1912.05752*.

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.

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

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*.

Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C Paulson. 2020. Isarstep: a benchmark for high-level mathematical reasoning. *arXiv preprint arXiv:2006.09265*.

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.

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 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. 2023. Generating mathematical derivations with large language models. *arXiv preprint arXiv:2307.09998*.

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*.

Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The lean theorem prover (system description). In *International Conference on Automated Deduction*, pages 378–388. Springer.

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*.

Stanislas Polu and Ilya Sutskever. 2020. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*.

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.

Jia Tracy Shen, Michiharu Yamashita, Ethan Prihar, Neil Heffernan, Xintao Wu, Ben Graff, and Dongwon Lee. 2021. Mathbert: A pre-trained language model for general nlp tasks in mathematics education. *arXiv preprint arXiv:2106.07340*.

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, Jordan Meadows, Lan Zhang, and André Freitas. 2023. Multi-operational mathematical derivations in latent space. *arXiv preprint arXiv:2311.01230*.

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*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

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

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.

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*.

Stephen Wolfram. 1999. *The MATHEMATICA® book, version 4*. Cambridge university press.

Yuhuai Wu, Albert Qiaochu Jiang, Jimmy Ba, and Roger Grosse. 2020. Int: An inequality benchmark for evaluating generalization in theorem proving. *arXiv preprint arXiv:2007.02924*.

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.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

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 Fine-tuning BERT and prompting GPT

Fine-tuning BERT. 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. Training took around a day of compute on an NVIDIA A100. 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 (Shen 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).

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

Prompting GPT. For each task, we engineer few-shot prompts with the aim to optimise static performance with respect to the `gpt-3.5-turbo` model using the OpenAI API. The results of prompt exploration are given in Tab. 4, where the selected design is highlighted in bold. We describe this prompt below:

"The following examples consist of a prompt (denoted by Prompt:) and a label (denoted by Label:).
`Prompt:` Sequence 1
`Label:` Label 1
`Prompt:` Sequence 2
`Label:` Label 2
`Prompt:` Sequence 3
`Label:` Label 3
`Prompt:` Sequence 4
`Label:` Label 4
Now given the following prompt, predict the label.
`Prompt:` Test Prompt"

The sequences all contain the same final annotation as the Test Prompt and are sampled from the training set. Additionally, an equal number of negative (Label: 0) and positive examples (Label: 1) are included as in-context examples, and these examples are shuffled. New lines are denoted by "\n". Perturbations are only applied to the Test Prompt and in-context examples are fixed to minimise examples' effect on generalisation. 200 random examples from the static test set *per subtask* (e.g. steps=2, integration) are used in the evaluation, which maps to 200 equivalent examples *per perturbation*. This totals around 4000 total examples *per GPT model*.

B Task-specific data format and sizes

The data generation algorithms output a derivation (Alg. 3) or expression (Alg. 2) in LaTeX and SymPy. Outputs are then adapted to fit specific tasks. For the described classification tasks, a single example consists of the reasoning sequence up to the final expression or equation (Fig. 4).

Constructing sequences. For the *Derivation Step Classification* task, a step consists of an equation and an annotation, as described in Fig. 1 and Fig. 4. 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.

Prompt Design	**GPT-3.5** (F1)	**GPT-4** (F1)
Derivation Step Classification (steps=2)		
No task description + random examples (2 pos, 2 neg)	61	83
Concise task description + random examples (2 pos, 2 neg)	50	83
No task description + same final operation examples (2 pos, 2 neg)	**68**	**90**
No task description + same final operation examples (3 pos, 3 neg)	68	87
Calculus Classification (differentiation)		
No task description (2 pos, 2 neg)	**55**	**55**
No task description (3 pos, 3 neg)	48	64

Table 4: Prompt designs trialled for the experiments.

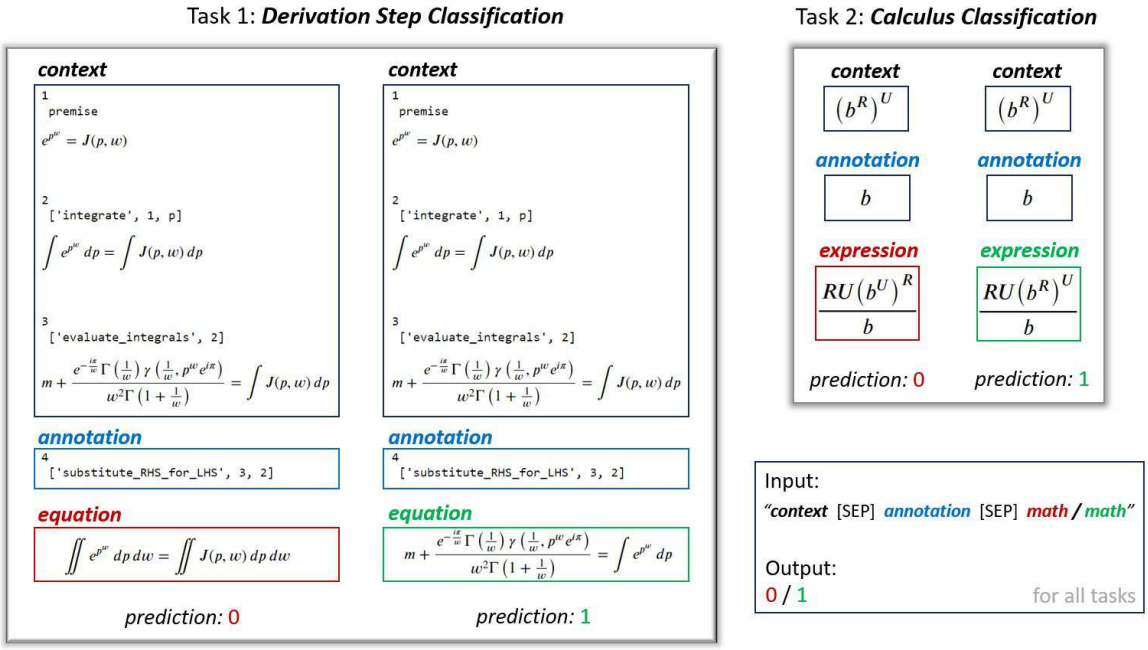

Figure 4: Sampled data from each binary sequence classification task. In short, a sequence containing reasoning ***context***, an instruction ***annotation***, and resulting ***math*** is input to a model. The model then predicts whether the math follows from the context and annotation, and if the sequence is mathematically coherent (1) or not (0).

Task	Training	Validation	Static Test	Perturbed Test
Derivation Step Classification				
2-steps	20K	5K	4K	4K
3-steps	20K	5K	4K	4K
4-steps	20K	5K	4K	4K
Calculus Classification				
integration	32K	8K	4K	4K
differentiation	32K	8K	4K	4K

Table 5: The number of examples considered by models during training, validation, and evaluation.

For *Calculus Classification*, an input sequence consists of a premise expression, a variable, and the result of either differentiating or integrating. The premise expression containing *at least two* variables is initially generated, a variable is randomly selected from the premise, 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. The three main components for each task are [SEP] separated. In the datasets for either task, this sequence is grouped with both the actual final equation and a number of negative equations. As a model encounters an example it is processed into

two sequences; one including the positive equation and another including the negative. Each sequence is then paired with the corresponding classification labels. Perturbations are applied to each test set and generate an equal number of perturbed examples. The *Derivation Step Classification* datasets include 41K evaluation examples *per derivation step count*. The *Calculus Classification* datasets include 52K evaluation examples *per operation*. This equates to ***227K total examples***. Tab. 5 describes the relevant sizes for the models.

C Supplementary Material for Qualitative Analysis

We can alternatively measure generalisability by examining the proportion of examples where predictions involving static sequences are correct, while predictions for mathematically equivalent perturbed sequences are incorrect. Defining an example to consist of a static sequence grouped with its perturbed equivalents, if a static prediction is correct while all perturbation predictions fail, this gives a strict measure of generalisability (denoted by G in Tab. 3) and complements previous analysis. These grouped examples allow examination of how well models understand each operator, and can highlight their weaknesses. We identify such weaknesses shared between GPT and BERT models and discuss clear dissimilarities in a more focused discussion in this section.

Why is R associated with generalisation failure for BERT but not for GPT? Prior analysis points to the premise renaming operator R as a useful point of comparison between fine-tuned BERT and few-shot GPT. Prompting GPT-3.5 by appending "Describe what function `renaming_premise` performs." to a static prompt (associated with GPT-3.5's generalisation failure) returns the following definition of R: *"the* `renaming_premise` *function is used to create a new expression or equation by assigning an existing expression or function to a new variable or function symbol."* This appropriate understanding persists even for perturbed prompts, and naturally extends to GPT-4. In contrast, further analysis (Appendix C) reinforces that BERT models do not share this out-of-distribution understanding. The main difference between R and all other operators is that it appears in sequences *without any reference to prior equations*. The substitution operations are the opposite of this (referencing the most equations of any operators), and *both* GPT-4 and BERT frequently fail to make correct predictions given this operator. On one hand, the operator with the *least referencing* is significantly associated with generalisation failure for BERT, but not GPT-4. On the other, the operator with the *most referencing* is not significantly associated with generalisation failure in either case, as all models are not effectively learning substitution in-distribution. BERT is dependent on more localised learning where the necessary semantics is expressed within a short text span during training, rather than a span that explicitly relates to other textual elements (*e.g.,* through regular reference). In other words, *a lack of explicit discourse relations that predictably vary with the ground truth obstructs models from learning latent relations that allow them to generalise.* However, the explicit relations can not be too complex (as with substitution). R lends itself to generalisation failure because it lacks structured discourse relations of the appropriate complexity for BERT (that others operators do not). R is simpler for GPT because of its varied exposure to structured text featuring such relations (*e.g.,* code) and obviously its relative size.

Focusing solely on BERT-related models in more depth, we consider (uncased) models trained on 3-step derivations. This number of steps closely reflects the average results over all step counts in Table 1. The **All** (perfect generalisation) and **Not P** (complete generalisation failure) columns of Table 6 (Appendix C) reinforce the relative generalisability gap 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 other operators).

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 further trained with scientific papers on MLM and NSP. MathBERT (Shen et al., 2021) was further trained from BERT on educational mathematical text, ranging from pre-k to graduate level difficulty. However, unlike BERT and SciBERT, MathBERT was trained to optimise performance on MLM over a large corpus. Fine-tuning generally overwrites representations learned from previous tasks (Mosbach et al., 2020), and MathBERT has likely forgotten those associated with NSP. The current classification tasks involve determining if *math context entails an expression/equation*, rather than predicting individual tokens as in language modelling. Next-equation prediction shares greater similarity with NSP than MLM, and we therefore attribute generalisability failures of MathBERT in this context to insufficient entailment pre-training. It has struggled with entailment before relative to other BERT models (Meadows et al., 2022).

Advantages of pre-training on structured scientific text. SciBERT differs from the other encoders due to a distinct focus on scientific papers written in LaTeX. This offers two benefits: **(1)** Mathematical elements seen by models are written in LaTeX, so exposure to LaTeX (during both MLM *and* NSP) provides natural advantage; **(2)** Scientific papers tend to be concise and logically structured. Text spans are carefully chained to reach conclusions, so exposure to papers during training may better teach models the concept of entailment and aid performance in related tasks.

	Static ±	**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 6: **Static ±** 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.

D Algorithm for Premise Generation

The *"Generate Premise Equation"* algorithm (Alg.2) aims to create a mathematical equation from a defined vocabulary of letters and operators. Specifically, the algorithm's process can be summarized as follows:

1. **Initialisation:** Symbols and mathematical operations are defined:
   - $\mathcal{S}$ represents all symbols from the vocabulary $\mathcal{V}$.
   - $\mathcal{R}_1$ comprises unary operations like Cosine, Sine, Exponential, and Logarithm.
   - $\mathcal{R}_2$ contains binary operations such as Addition, Subtraction, Multiplication, etc.
2. **Base RHS Construction:** Depending on a randomly chosen arity (either 1 for unary or 2 for binary):
   - If arity = 1, the RHS is built by applying a random unary operation on a random symbol.
   - If arity = 2, the RHS is constructed using a binary operation on two distinct random symbols.
3. **Complexifying the RHS:** A random complexity value is selected from 0 to $\mathcal{C} - 1$. For each iteration up to the chosen complexity, the RHS's complexity is increased by applying either a unary operation on the current RHS or

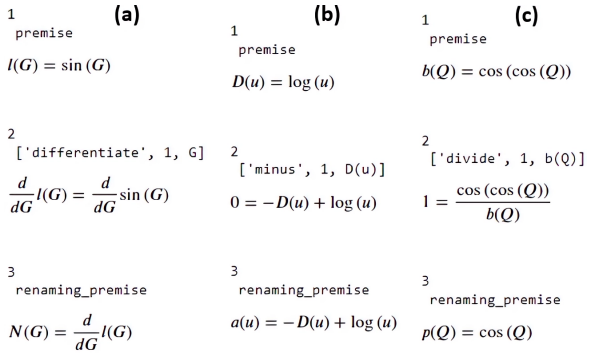

Figure 5: Three examples of the total 15 where both SciBERT and MathBERT correctly classify *unperturbed* examples (as shown), but incorrectly classify all perturbed examples.

a binary operation between the current RHS and another random symbol.

4. **LHS Construction:** The LHS is then formulated as a function of the free symbols present in the RHS.

5. **Equation Formation:** Lastly, an equation, termed *premise*, is crafted using the finalized LHS and RHS.

In essence, this algorithm dynamically produces a mathematical equation whose intricacy varies depending on the randomly chosen operations and the selected complexity.

E Algorithm for Derivation Generation

Algorithm 3 relies on Algorithm 2 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.

Runtime. To calculate the time taken to generate a derivation, we sample a number of derivation lengths (*i.e.,* number of equations) from a Gaussian $N(6.5, 3)$ truncated between 4 and 9 inclusive. It took 71 minutes to generate 100 derivations, with an average length of 6 (*i.e.,* 0.7 min/derivation, 7 seconds per step), on a mid-range laptop CPU. In addition to the time taken for SymPy to perform complex calculus operations, it takes some time to generate valid derivations that do not repeat steps and fit within our given parameters, hence hundreds of steps may fail for a given derivation.

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*, 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. The exact form of the algorithm used to generate data for this paper is available in the linked repository on the first page.

In more formal detail, the mechanics of Algorithm 3 are as follows:

1. **Procedure Step**: This subroutine generates a single step in the derivation.
   - Sets of equations, operations, and other relevant elements are initialised from the dataset $\mathcal{D}$.
   - Based on probability parameters, the arity of the operation (either 0, 1, or 2) for this step is determined.
   - Depending on the chosen arity:
     - Arity 0: The equation and annotation for this step are directly chosen from the set $\mathcal{R}_0$.
     - Arity 1: An operation from $\mathcal{R}_1$ and an equation from the dataset are chosen to form the new equation.
     - Arity 2: An operation from $\mathcal{R}_2$, an equation from the dataset, and another element are selected to shape the equation.
   - If the formed equation is deemed valid

Algorithm 2 Generate Premise Equation

Assumes a global vocabulary of letters, $\mathcal{V}$ and operators *e.g.,* cos, sin, etc. Accepts a complexity hyperparameter $\mathcal{C}$ that determines the maximum tree depth of the premise RHS.

```
procedure PREMISE(C)
    S ← [Symbol(v) for v in V]
    R_1 ← [Cos, Sin, Exp, Log]
    R_2 ← [Add, Minus, Times, Power, Divide, Differentiate, Integrate]
    arity ← random.choice([1,2])
    if arity = 1 then
        R ← random.choice(R_1)
        S ← random.choice(S)
        RHS ← R(S)
        LHS ← random.choice([s for s in S if s ≠ S])
    else if arity = 2 then
        R ← random.choice([r for r in R_2 if r not in [Differentiate, Integrate]])
        S_1 ← random.choice(S)
        S_2 ← random.choice([s for s in S if s ≠ S_1])
        RHS ← R(S_1, S_2)
        LHS ← random.choice([s for s in S if s not in [S_1, S_2]])
    end if
    complexity ← random.choice(range(C))
    for i ∈ range(complexity) do
        arity ← random.choice([1,2])
        if arity = 1 then
            R ← random.choice(R_1)
            RHS ← R(RHS)
        else if arity = 2 then
            R ← random.choice(R_2)
            S ← random.choice(S)
            RHS ← R(RHS, S)
        end if
    end for
    LHS ← Function(LHS)(*tuple(RHS.free_symbols))
    premise ← Eq(LHS, RHS)
    return premise
end procedure
```

through certain checks it is returned; otherwise, None is returned.

2. **Main Derivation Loop**: This section assembles the derivation.
   - The initial step of the derivation is generated using Algorithm 2.
   - A pre-defined target length L_i describes approximately the number of times the *Step* procedure is invoked to add new steps.
   - The full derivation is extracted from the accumulated steps.
   - The loop breaks when the derivation reaches a desired length L_f, where $L_f \geq L_i$.

To summarize, the algorithm iteratively constructs a derivation of mathematical equations, where each step is shaped by a series of operations determined by specific probabilities and conditions. It is given on the following page.

Algorithm 3 Generate Equational Reasoning

```
procedure STEP(𝒟, p_0, p_1, p_2, p_h, p_r, p_e, p_c, p_s)
    D ← [i[0] for i in 𝒟]
    A ← [i[1] for i in 𝒟]
    ℛ_0 ← [Premise] + [RenamingPremise]×p_r
    ℛ_1 ← [Cos, Sin, Exp, Log, Expand] + [EvaluateDerivatives, EvaluateIntegrals]×p_e
    ℛ_2 ← [Add, Minus, Times, Divide, Power] + [Differentiate, Integrate]×p_c
            + [SubsLHSForRHS, SubsRHSForLHS]×p_s
    elements ← numbers, variables, and subexpressions from D
    arity ← random.choice([0]×p_0 + [1]×p_1 + [2]×p_2)
    if arity = 0 then
        R ← random.choice(ℛ_0)
        equation ← R
        annotation ← R.__name__
    else if arity = 1 then
        R ← random.choice(ℛ_1)
        e_1 ← random.choice(EquationDistribution(D, p_h))
        equation ← R(e_1)
        n ← D.index(e_1)
        annotation ← [R.__name__, n + 1]
    else if arity = 2 then
        R ← random.choice(ℛ_2)                          ▷ R depends on the length of D
        e_1 ← random.choice(EquationDistribution(D, p_h))
        e_2 ← random.choice(elements)                   ▷ e_2 will vary depending on R
        equation ← R(e_1, e_2)
        n ← D.index(e_1)
        annotation ← [R.__name__, n + 1, e_2]
    end if
    if equation is valid then                           ▷ validity depends on various checks
        return equation, annotation
    else
        return None
    end if
end procedure
while True do
    𝒟 ← [(Premise(𝒞), "premise")]                       ▷ generate first step using Algorithm 2
    while len(𝒟) < L_i do                               ▷ L_i is an initial length of the derivation
        step ← Step(𝒟, p_0, p_1, p_2, p_h, p_r, p_e, p_c, p_s)
        if step is not None then
            𝒟.append(step)
        end if
    end while
    derivation ← ExtractDerivation(𝒟)
    if len(derivation) = L_f then                       ▷ L_f ≥ L_i is the desired length of the derivation
        break
    end if
end while
𝒟 ← derivation
```