# Modeling Mathematical Notation Semantics in Academic Papers

**Hwiyeol Jo**[*] NAVER, Clova AI hwiyeolj@gmail.com

**Dongyeop Kang**[†] Univ. of Minnesota dongyeop@umn.edu

**Andrew Head** **Marti A. Hearst** Univ. of California, Berkeley {andrewhead,hearst}@berkeley.edu

## Abstract

Natural language models often fall short when understanding and generating mathematical notation. What is not clear is whether these shortcomings are due to fundamental limitations of the models, or the absence of appropriate tasks. In this paper, we explore the extent to which natural language models can learn semantics between mathematical notation and their surrounding text. We propose two notation prediction tasks, and train a model that selectively masks notation tokens and encodes left and/or right sentences as context. Compared to baseline models trained by masked language modeling, our method achieved significantly better performance at the two tasks, showing that this approach is a good first step towards modeling mathematical texts. However, the current models rarely predict unseen symbols correctly, and token-level predictions are more accurate than symbol-level predictions, indicating more work is needed to represent structural patterns. Based on the results, we suggest future works toward modeling mathematical texts.

## 1 Introduction

With the enormous growth of academic publishing, there is increasing interest in the area of scholarly document analysis in NLP (Chandrasekaran et al., 2020; Beltagy et al., 2019). Many academic papers use mathematical notation, both in formulas and in describing components of algorithms, as seen in '$\alpha$ is the learning rate', and '240 × 240 pixel image'. However, despite the great advances in pretrained language models such as BERT (Devlin et al., 2019), they are still unable to analyze mathematical notation reliably (Andor et al., 2019). Similarly, in our experiments (§4.3), pretrained models show very poor performance (9%), when compared with the traditional N-gram based models (19%). This empirical result tells us that BERT's pretraining method barely incorporates the modeling of mathematical notation. Although we find that fine-tuning through masked language modeling on academic text increases the model performance (by 48%), the accuracy is still low for an application.

We cast the problem of mathematical notation semantics as one of predicting the right notation given the context – notation prediction has the advantage of simplicity. Motivated by an academic paper authoring application (Head et al., 2020), we propose two new target tasks that can make use of mathematical notation prediction: notation auto-suggestions and notation consistency checks.

**Notation auto-suggestions**: Some symbols are used conventionally in a given domain. For instance, the symbol $\alpha$ is used conventionally in deep learning papers as learning rate. A tool for suggesting notation could learn about conventions from usages across many papers, and make suggestions for appropriate notation given its context.

**Notation consistency checks**: When writing technical papers, authors develop the ideas during the process of writing. This can lead to inconsistency in notation. For instance, a concept might be referred to as $D$ to mean *delta* in Equation 1 in the paper, and then later the authors may use $D$ to mean *document* in Equation 3. A notation verification tool could check the consistency of notation usage across the paper and warn the author when the different uses of the symbol might be in conflict.

In addition to the tasks, we propose an approach to fine-tuning BERT language models to represent mathematical notation with a level of accuracy that begins to approach that needed for these kinds of real-world applications. As shown below, our method achieves top-1 token-level accuracy of 61% and 74% for notation auto-suggestion and notation consistency checks, respectively.[1]

[*] Work carried out at Seoul National University

[†] Work carried out at UC Berkeley

[1] The codes and dataset are available at

*Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3102–3115
November 7–11, 2021. ©2021 Association for Computational Linguistics

Our contributions can be summarized as follows:

- We propose two notation prediction tasks to test models' mathematical semantics understanding.
- We then present a fine-tuned model MathPredictor for the tasks, showing +12%, +16% performance gain compared to BERT for the two tasks, respectively.
- We show analyses of the results that suggest that the performance does not largely depend on the global context. We also report evidence that our method can learn common notation usage patterns.

## 2 Related Work

**The use of mathematical notation in texts.** Mathematical notation is integral to academic discourse. The symbols in a single academic paper often number in the hundreds (West and Portenoy, 2016; Greiner-Petter et al., 2020). A single instance of notation can be simple, as in the case of a single-letter symbol, or complex, as in an equation composed of dozens of symbols.

In a given text, the prose and notation are closely related and inter-referential. Studies of reading behavior reveal readers diverting their focus from notation to text and vice versa (Zanibbi and Blostein, 2012; Kristianto et al., 2014; Kohlhase et al., 2018). Definitions of notation appear close to the notation itself; Wolska and Grigore (2010) estimated that as many of 70% of symbols are defined in the same paragraph that the symbol is introduced. This close relationship between prose and notation suggests that language models may be able to select notation to suit the context it appears in.

**Modeling mathematical notation.** Neural networks have been used widely to predict textual tokens from their context, with simple models such as word2vec (Mikolov et al., 2013) and high-capacity transformers (Devlin et al., 2019). However, could similar techniques be used to predict mathematical notation given the textual context it appears in? Recent research has explored this possibility in various tasks such as type inference in mathematical statements (Rabe et al., 2020) by using Transformer architecture (Vaswani et al., 2017), topic-sensitive equation generation (Yasunaga and Lafferty, 2019), superscript disambiguation (Youssef and Miller, 2018) with recurrent neural networks, and mathematical information retrieval (Greiner-Petter et al., 2020) using word2vec algorithms. These results show that the recent advanced models could be used for modeling notation.

A complementary line of research takes notation as input, and attempts to predict text to describe the notation. In recent work, Abekawa and Aizawa (2016) retrieved a relevant paragraph to a given query, Kang et al. (2020) and Alexeeva et al. (2020) extracted symbol descriptions from text, and Madisetty et al. (2020) detected symbol descriptions in math equations. In contrast to prior work, our model takes text as input and *predicts* appropriate mathematical notation.

**Text-based representations of notation.** Our method also differs from prior work on modeling mathematical notation in the granularity of its textual predictions: the model predicts tokens of equations in LaTeX representation. In this way, our method simultaneously supports the prediction of simple symbols comprising single tokens (e.g., "$x$"); composite ones made up of multiple tokens ("$x_i$"), and symbols with accents ($\bar{x}$) and styles ("$\mathbf{x}$"), given that symbol relationships, styles, and accents are all declared explicitly in LaTeX.

This paper chooses text-based representations of notation due to the simplicity of the approach; such representations requires minimal changes to existing successful natural language models like BERT, allowing attention to be devoted to the development and evaluation of an appropriate task.

Contemporaneously with our work, Peng et al. (2021) proposed a method for pre-training a model to predict mathematical equations by encoding tree structures of equations as well as their surrounding text. By contrast, we apply a fine-tuning approach since we hypothesized that the power of pre-trained BERT on language understanding will be important for modeling notation. Furthermore, while Peng et al. (2021) focused on predicting mathematical equations, our work predicts not only mathematical equations but also mathematical notation such as numbers, latex macros, letters, symbols, and math operators, making it more appropriate for academic authoring applications.

## 3 Proposed Method: MathPredictor

We design a method to learn the meaning of math notation grounded in the surrounding text. This requires good representations of standard text, mathematical notations, and their combination. To

http://github.com/HwiyeolJo/MathPredictor

| Types | Examples | #-Uniq |
|---|---|---|
| `Letter` | n, m, SHA, model, loss, x .. | 16K |
| `Number` | 218, 00, 4k, 2K, 90., 2cm .. | 234 |
| `OP&Symbol` | $\alpha$, $\theta$, $\leq$, $\times$, arccos, %, $\exists$ .. | 271 |
| `LaTeX Macros` | \top, \text, \mathcal \quad,<br>\bf, \rm, \underline, \em .. | 562 |

**Table 1:** Examples of notation tokens. We report the unique number of notation tokens in the training data. #-Uniq means the unique number of notation tokens.

achieve this, we use a pre-trained language model BERT (Devlin et al., 2019) for the standard text representation and fine-tune it *only* on notation.

Our model takes raw LaTeX files as input, which have some markedly different characteristics than general natural language text input. In this section we define the notations that our system recognizes (§3.1), and how it is tokenized (§3.2). Then we describe our model MATHPREDICTOR and its training and inference procedure (§3.3).

### 3.1 Notation Type Definition

We call every occurrence of tokens between \$ signs in the raw LaTeX files *mathematical notation tokens*. Notation tokens consist of one or more *letters*, *numbers*, *math operators and/or symbols* (including Greek letters), and LaTeX's default *macros*. Table 1 shows some example of notation tokens for these four categories, along with their unique number in the training set (the dataset will be described in §4.1). The algorithm for tokenizing into these categories assumes tokens are separated by blank space, and works as follows:

**Numbers:** these must begin with a digit followed by any number of digits commas, and periods, and ending with digits or English letters before the next blank space. This allows for covering large integers and decimal numbers and numbers with measurement (e.g., cm).

**LaTeX macros:** if the token starts with \(backslash), the following letters up until blank space are grouped into a LaTeX macro, with several manually determined exceptional cases such as greek letters ($\alpha$ (`\alpha`)) to be grouped as math symbols instead.

**Letter notations:** The letter notations begin with English characters, and all following characters up to a blank space.

What remains are considered as **math operators and symbols**. We did extra checking on the final groups manually. The details on the manual process are described in Appendix A.

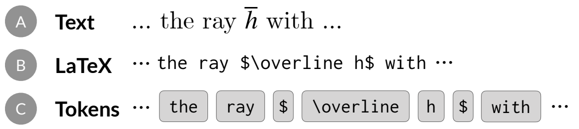

A
Text
... the ray $\overline{h}$ with ...
B
LaTeX
··· the ray $\overline h$ with ···
C
Tokens
··· the ray $ \overline h $ with ···

**Figure 1:** Illustration of the tokenization process. After updating the vocabulary with the expanded LaTeX macros, individual LaTeX expressions can be successfully tokenized.

### 3.2 Tokenization

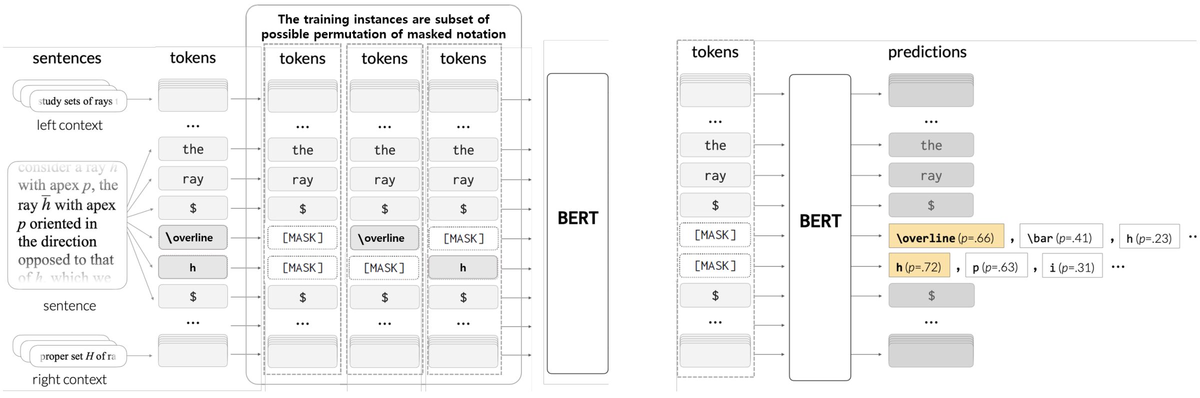
We tokenize raw LaTeX using BERT's default WordPiece tokenizer (Wu et al., 2016). However, we observe that BERT's tokenizer splits LaTeX macros like `\mathbf` or symbols `\alpha` into incorrect sub-words. For example, the macro `\overline` is split to multiple sub-tokens \, `over`, and `##line` by the original BERT's tokenizer. To address this problem, we extract entire macro vocabularies from training data by using regular expressions (future work can make use of a LaTeX parser). After adding the extracted macros, the original BERT's vocabulary size increases from 28,996 to 31,647. Figure 1 illustrates the output after tokenization.

The final tokens are then fed to a model following BERT's standard encoding scheme: adding the special token `[CLS]` at the beginning of sentence and `[SEP]` at the end. After the `[SEP]` token, we add `[PAD]` tokens to match the maximum token length. `[MASK]` tokens are used for masking target tokens to predict the target sentence, which will be further described in the next section. After adding the special tokens, we truncate the input text to 512, which is the maximum token length of original BERT.

### 3.3 Training and Inference

Our model MATHPREDICTOR is fine-tuned from the pre-trained language model BERT to predict the masked tokens of *math notation* in academic papers. Figure 2 illustrates how MATHPREDICTOR works at training and inference. We mask only individual tokens of the mathematical notation, not any non-notation words. We hypothesize that this targeted training allows the model to learn a better relationship between text and math notation than standard BERT; this is verified in our empirical evaluation (§4.4).

The goal of this method is to correctly predict



**(a)** Training

**(b)** Inference

**Figure 2:** The illustration of the proposed method. It encodes context (left and/or right sentences) and the target sentence where each token of notation in the target sentence is masked (`[MASK]`). At training, we permute the input sequences (dotted boxes) with random probability $p$ in order to learn the structure of notation and then train BERT by using these representations of the sentences. As a result, the training instances are subset of the permutations. At inference time, the masked token is predicted with likelihood scores.

the masked tokens ('$h$', '$p$', '\overline') in the target sentence by looking at surrounding text in the target sentence as well as the context sentences. The context sentences can be either sentences to the left of the target sentence for the auto-suggestion task or sentences to both the left and right of the target sentence for the consistency checking task. During training time, we optimize cross-entropy over notations only. At testing time, the model outputs the top-k candidates for each notation token using maximum likelihood and calculates mean reciprocal rank scores (Voorhees, 1999).

**Permutation over notation tokens.** The model should learn the connections between notation tokens. For instance, a LaTeX macro `\overline` should be followed by letters or symbols (e.g., $h$) in order to build a fully correct symbol ($\overline{h}$). Thus, inspired from Rabe et al. (2020) we add additional training instances that partially mask the notation with random probability $p$. As a default, each token is masked with the probability of 0.9. This technique helps the model learn mathematical notation via the other surrounding notations and augments the number of training examples.

**Model choice.** Our model predicts math notation in the target sentence in a non-autoregressive manner of BERT (Devlin et al., 2019). This means that all of the notations in a given context are predicted at once rather than sequentially. An alternative approach is to use autoregressive language modeling such as GPT2 (Radford et al., 2019). However, this takes too much time to decode each token and is also less suitable for our scenario of writing assistant applications, since authors rarely write complex research papers in a strictly sequential order.

**Notation length constraint.** In some cases, sentences have very long sequences of mathematical notation, as seen in this example: "Let the data source be $ Y \text { \in } \lbrace Y _ 1 , \dots , Y _ m \rbrace $." In this case, the model needs to predict a total of 18 notation tokens between $ signs, which is quite difficult. Thus, we restrict the maximum number of masked tokens (i.e., notations) in a sentence to be less than **10** to alleviate the level of task difficulty. For reference, we present the performances as the length constraint changes in Appendix B.

**Larger context modeling.** A research paper typically spans many pages. Sometimes, notation can be defined early in the paper and then re-used later in many sentences, paragraphs, or even sections away.

Therefore, we test the hypothesis that the fine-tuned model needs to understand not only sentences immediately surrounding the occurrence of mathematical notation, but relevant sentences at any positions in the paper. We refer to this as *global context*, and to the surrounding context before and/or after the target sentence as *local context*.

Our training is based on the pre-trained BERT model, which cannot encode the context longer than its maximum token length which is 512. To remedy this, recently researchers have developed long-encoder BERT such as the longformer model (Beltagy et al., 2020), but its inference time is too slow to be used as a practical feature for real-time writing assistant.

Instead, we extend the fine-tuned model to encode global context by simply averaging sentence representations in the global and local context together. Specifically, we first encode each sentence into a vector and get $v_{S_1} \cdots v_{S_N}$, where the sentence representation is made by `[CLS]` token of the last hidden layers. Then, we concatenate sentence vectors in the global and local context by averaging: for example, assume that the model is designed to encode 7 previous sentences $S_{n-7}, \cdots, S_{n-1}$ from the target sentence $S_n$ as the local context. For the global context, we average the previous vectors from the local context and then concatenate the averaged vector to the input token embedding. That is, we averaged the `[CLS]` vectors of $S_1, \cdots, S_{n-8}$ and concatenate it with the vectors of local context, which is $S_{n-7}, \cdots, S_{n-1}$ and then predict the notation in the target sentence $S_n$. By doing so, the model can use all the previous sentences.

Likewise, in notation consistency checks with left 3 and right 3 context sentences, in addition to the averaged vector of the previous sentences, the other next sentences out of local context are averaged and concatenated to the last of the input token embeddings. As a result, the input will be $avg(\texttt{[CLS]}_{S_1} \cdots \texttt{[CLS]}_{S_{n-4}})$ $S_{n-3} \cdots S_n \cdots S_{n+3}$ $avg(\texttt{[CLS]}_{S_{n+4}} \cdots \texttt{[CLS]}_{S_N})$. We call these method as full context model.

## 4 Experiments and Evaluation

This section evaluates how well our model performs at mathematical notation prediction compared to several strong baselines. We first describe the evaluation dataset (§4.1), the baseline models (§4.2), and the notation prediction tasks (§4.3). We then investigate the following research questions:

R1 (§4.4): How well does our model predict when compared to the baselines for the two prediction tasks?

R2 (§4.5): Does the model simply memorize notations in context or does it learn domain-conventional patterns from other papers?

R3 (§4.6): Which types of notations is the model most able to predict?

R4 (§4.7): How well does the model perform when evaluated at the sentence level?

R5 (§4.8): How well does the model perform at the document-level (qualitatively)?

### 4.1 Dataset

For our experiments, we use the S2ORC (Lo et al., 2020) dataset which contains 12.7 million full text of research papers; many of these papers contain mathematical notations written in LaTeX format.

From S2ORC, we randomly sub-sample 1,000 papers as our experimental dataset. This dataset size is similar to the previous works (Aizawa et al., 2014; Abekawa and Aizawa, 2016) and is necessary due to computational constraints.

We identify sentence boundaries with NLTK's sentence tokenizer and tokenize words with BERT's WordPiece tokenizer (Wu et al., 2016). The resulting dataset has on average 223.2 sentences per document and 20.3 tokens per sentence.

We split the data into train, validation, and test set with the ratios of 80%, 10%, and 10%. To prevent the split data from being biased, we tested baseline models such as random prediction and n-gram models on the data split, and then selected the split that showed average performance. After pre-processing the test set, there are 14K tokens to be predicted, which is 3.05 tokens per symbol and 4.84 tokens per sentence. On average there are 1.59 mathematical notations included in a sentence.

Academic papers contain some frequent non-text entities that are not relevant to the mathematical notation task, and so we replace these with placeholder terms. These include converting citations (e.g., Author et al.) to `CITATION`, section references (e.g., §4) to `SECTION`, long equations to `EQUATION`, tables to `TABLE`, and figures to `FIGURE`. This preprocessing step reduces unknown vocabularies, mitigating noise that might prevent encoders from understanding the text.

### 4.2 Baseline Models

As described in §3.3, we choose bert-base-cased (Devlin et al., 2019) as a base encoder for MathPredictor. Note that we use the cased version because in our tasks upper-cased (e.g., $N$) and lower-cased (e.g., $n$) notations can have distinct mathematical meanings.

In addition to the pre-trained baselines, we also build strong baselines by fine-tuning the original pre-trained baselines on our dataset using the standard masked language model training on both notation and text. Other BERT variations such as SciBERT (Beltagy et al., 2019) fine-tuned on scientific papers and RoBERTa (Liu et al., 2019) are also used as baseline models. Additionally, stan-

| | Suggestion | | | Consistency | | |
|---|---|---|---|---|---|---|
| | Top1 | Top5 | MRR | Top1 | Top5 | MRR |
| Random | 3.3 | 14.1 | - | 3.6 | 15.1 | - |
| 4-gram | 18.8 | 28.5 | - | - | - | - |
| BERT | 9.0 | 18.8 | .146 | 13.8 | 28.3 | .215 |
| BERT(FT) | 48.3 | 66.1 | .568 | 57.8 | 75.4 | .658 |
| SciBERT | 15.19 | 26.2 | .207 | 16.6 | 26.6 | .216 |
| SciBERT(FT) | 48.8 | 68.8 | .579 | 58.6 | 76.7 | .669 |
| RoBERTa | 0.5 | 1.5 | .011 | 1.7 | 3.6 | .029 |
| RoBERTa(FT) | 21.9 | 33.1 | .277 | 32.8 | 45.8 | .393 |
| MathPred | 57.4 | 65.4 | .613 | 71.7 | 77.7 | .746 |
| MathPred(FT) | **60.5** | **71.3** | **.657** | **73.5** | **80.0** | **.767** |
| w/ FullContext | 55.7 | 68.7 | .620 | 72.2 | 79.8 | .758 |

**Table 2:** Performance comparison on notation auto-suggestion and consistency checking tasks. FT means fine-tuning the model through masked language modeling on notations and words using our dataset. w/ FullContext means using full global context with MathPredictor.

dard baselines such as random selection and n-gram based models are also presented.

We set the hyperparameters to the defaults by Huggingface's training scripts (Wolf et al., 2020), with the exception of setting a learning rate of $5e-6$ and early-stopping using validation loss. Additional details for baseline training, hyperparameter settings, and computing resources can be found in Appendix C.

### 4.3 Tasks

**Notation auto-suggestions.** To simulate the auto-suggestion task, this evaluation attempts to automatically suggest notation by using the text of the target sentence as well as sentences to the *left* of the target sentence. Every masked token is predicted from among the vocabulary of the tokenizer, and the top-k tokens are chosen as the final candidates. Note that the evaluation in §4.4 reports token-level top-k accuracy with mean reciprocal rank (MRR) scores and we report notation-level and sentence-level accuracies in §4.7.

**Notation consistency checks.** The consistency checking task verifies whether notations are used consistently within a paper. To simulate it, we evaluate the use of notation in the target sentence with respect to the context it is found in. Therefore, the context used for consistency checking is given as sentences to the *left* and *right* of the target sentence.

Then, we design a prediction task that chooses the gold notation among candidate choices. The negative notations are produced by replacing each gold notation token with a random token of the same notation type (see Table 1) of the gold token. For example, the letter token $n$ is replaced by other letter tokens like $p$, and the symbol token $\alpha$ is replaced by other symbol tokens like $\beta$.

| | |
|---|---|
| Easy | ··· We use a ring dimension $n = 8192$ with two plain text moduli $t^{(j)}$. Each coefficient modulus $n = 8192$ is decomposed into four 64-bit moduli for efficient use of FV-RNS. |
| Challenge | ··· In scoring boardgames like Scrabble, swing, a state transition of advantage during the game progress is considered as successful shoot, and game length as attempt respectively. Let $S$ and $N$ be the average number of swings and the game length, respectively. |

**Table 3:** Examples of the easy set and the challenge set. The easy set has the target notations in the context sentences whereas the challenge set does not. The underlined sentences are used as context and the blue-colored symbols with gray backgrounds are the target. These examples are from the auto-suggestion task.

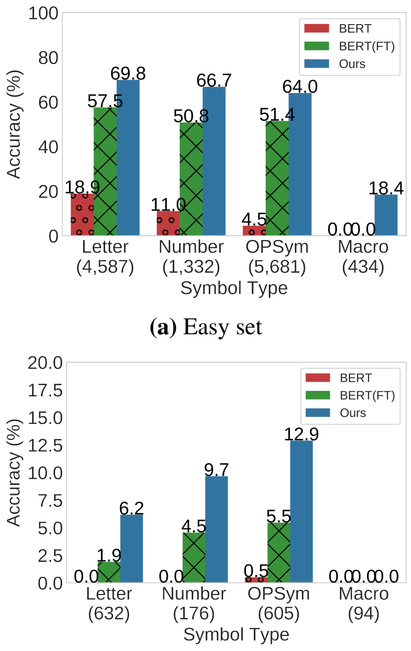
| | Suggestion | | Consistency | |
|---|---|---|---|---|
| | Easy | Chal. | Easy | Chal. |
| BERT | 9.99 | 0.26 | 15.09 | 0.12 |
| BERT(FT) | 52.32 | 3.38 | 59.27 | 3.44 |
| MathPred(FT) | **66.97** | **7.62** | **77.72** | **6.38** |
| #-samples | 12,364 | 1,511 | 12,382 | 826 |

**Table 4:** Top-1 accuracy of notation auto-suggestion and consistency checks on the easy set and challenge set. Note that the total sum of samples are different due to the different window sliding.

### 4.4 Notation Prediction Tasks

Table 2 shows the top-1, top-5 accuracy and MRR scores of our method in comparison with other BERT variations and other baselines. The results show that the original BERT baselines are poor at the prediction tasks. BERT that were fine-tuned on our datasets (BERT(FT)) show relatively better results. However, our method shows significant improvements over all of the baseline models, showing that MathPredictor is particularly optimized to learn notation-specific semantics from text. Although the top-1 accuracy might not be enough, the top-5 accuracy is promising when we imagine that the application aids the writer, presenting possible notation candidates.

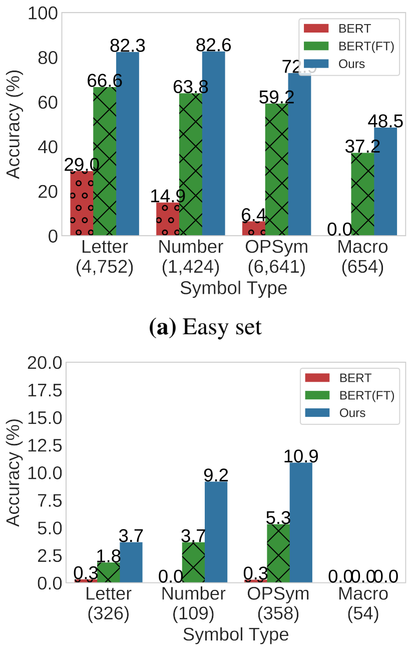
Modeling longer contexts (FullContext) degrades the performance on both tasks. We conjecture that the model is confused by additional global context that contains many other notations and/or



**(a)** Easy set

**(b)** Challenge set

**Figure 3:** Top-1 accuracy performance in notation auto-suggestion on notation types. (Number) denotes the total number of notation tokens in the test set. “Ours” refers to MATHPREDICTOR.



**(a)** Easy set

**(b)** Challenge set

**Figure 4:** Top-1 accuracy performance in consistency check on notation types. (Number) denotes the total number of notation tokens in the test set.

current averaging method barely distinguish them.

In Appendix D we provide ablation studies of model performances on different context sizes.

### 4.5 Assessing the Role of Local Occurrences

In this section, we ascertain whether MATHPREDICTOR predicts notation tokens based on seeing them in the surrounding context, or if it is learning conventions of notation from other papers in the training set (such as learning that $\alpha$ is conventional for learning rate). To measure this effect, we split the test samples in the two notation prediction tasks into two sets: If the notation token is present in the context sentences, we place it into `Easy` set. If not, we place it into `Challenge` set. Table 3 shows an example of each type.

Table 4 shows the performance on the easy and challenge sets. Although the original BERT shows poor performance on the easy set, the fine-tuned BERT seems to learn notations through masked language modeling. On the other hand, MATHPREDICTOR makes further improvements, by 14.6% for the auto-suggestion task and 17.5% for the consistency task.

However, the performance in the challenge set (7.6% for auto-suggestion and 6.4% for consistency) is much lower than the easy set (67% and 78%, respectively). This indicates that learning the domain-conventional patterns of notation usages is still relatively challenging compared to memorizing notations from given context.

### 4.6 Comparison of Notation Types

The next research question is which mathematical notation types (Table 1) are most successfully predicted. Figure 3 shows notation suggestion performance over different types of notations. Results for the other prediction task, notation consistency, have similar behavior and appear in Figure 4. Interestingly, all of the models have difficulty predicting LaTeX macros.

BERT cannot predict any of the operators or symbols in the challenge set. The fine-tuned BERT also struggles to learn the other forms of notation. On the other hand, MATHPREDICTOR predicts unseen notation better, achieving 6.2% for letters, 9.7% for numbers, 12.9% for symbols and operators in the challenge set. However, MATHPREDICTOR still fails to predict LaTeX macros, meaning that these patterns are difficult to learn. For example, the model needs to learn when authors use \mathbf for their stylistic choices.

| | Suggestion | | Consistency | |
|---|---|---|---|---|
| | Notation | Sent. | Notation | Sent. |
| BERT | 12.05 | 6.44 | 18.64 | 10.80 |
| BERT(FT) | 37.87 | 28.23 | 45.41 | 33.72 |
| SciBERT(FT) | 40.57 | 30.70 | 50.80 | 40.04 |
| MathPred(FT) | **45.11** | **37.11** | **57.20** | **48.56** |
| #-samples | 5,672 | 2,888 | 4,711 | 2,769 |

**Table 5:** The comparison of notation-level and sentence-level top-1 accuracy in both tasks. The total number of tokens can be different because of pre-defined vocabularies in tokenizer.

| **Multi-tokens in notation** | |
|---|---|
| ..Thus, the procedure is worth trying in this range of [MASK] [MASK] [MASK] [MASK] [MASK] [MASK] [MASK] [MASK].. | |
| Gold: | \hat { p } _ { com ##m } ; $\hat{p}_{comm}$ |
| BERT: | $ $ $ $ $ $ $ $ |
| BERT(FT): | **\hat {** 1 **}** , , $ $ **}** |
| Ours: | **\hat { p } _ { com ##m }** |
| **Multi-notations in sentence** | |
| ..that is they earn [MASK] [MASK] [MASK]/hour for class-1 VMs, [MASK] [MASK] [MASK]/hour for class-2VMs, and [MASK] [MASK] [MASK]/hour for class-3 VMs.. | |
| Gold: | 0 . 08 / 0 . 16 / 0 . 32 |
| BERT: | $ $ $ / $ $ $ / $ $ = |
| BERT(FT): | **0 . 08** / **0 .** 10 / **0 .** 08 |
| Ours: | **0 . 08** / **0** = **16** / **0 . 32** |

**Table 6:** Example of notation-level and sentence-level predictions. Correctly predicted tokens are shown in bold blue, and incorrectly predicted tokens are in red. our method shows better performance than the baselines, but fails to predict the notation tokens perfectly.

### 4.7 Notation- and Sentence- level predictions

Although our method shows competitive performance on token-level predictions, a more realistic assessment checks is performance over multiple tokens in a sequence of notation tokens and over multiple notations in the full sentence.

The only difference between this evaluation and the token-level experiment in §4.4 is that the model prediction is marked as correct only if every token in the individual notation or in the full sentence is correct.

Table 5 summarizes the results, and shows that success at multi-token notation-level and sentence-level is more difficult than at the token-level. For example, in Table 6 (top), the fine-tuned BERT is partially correct at notation-level but the model shows incorrect results in notation- and sentence-level evaluations. MATHPREDICTOR shows better performance, but it also fails to predict multi-notations in the sentence (see Table 6 (bottom)).

Interestingly, when comparing token-level accuracy (Table 2) and notation/sentence-level accuracy (Table 5), BERT shows better performance at notation/sentence-level prediction than token-level prediction, whereas the other models show the converse. This implies that the original BERT has more structural consistency. Therefore, this notation/sentence-level evaluation is important to test the models' ability to predict the structure of mathematical notations.

### 4.8 Qualitative Results: Full Paper

To simulate a real-world scenario, we run our model over sentences of a full paper. The model sequentially predicts notation tokens for each target sentence by looking at its local context, concatenating the prediction results to the next prediction, and repeating the process until the last sentence. From the test set, we select a paper which has many simple notation tokens to see the potential use case of MATHPREDICTOR.

Table 7 shows a paragraph extracted from the paper with prediction results by models. Similar to the previous empirical results, the original BERT shows very poor performance. The fine-tuned BERT and MATHPREDICTOR perform well, and interestingly, our model shows more consistency in using notation. In more detail, the fine-tuned BERT's predictions on the sentence "the length can be written as $N$, where $N$ is the probability of $m$ symbol" shows that it correctly predicts $N$ based on the previous sentence, but it does not predict the values for $m$ or $Y$ correctly.

On the other hand, MATHPREDICTOR correctly predicts the relationship between $m$ and $Y$. This example suggests that our method can learn some connections between symbol usages–possibly from the previous/surrounding sentences or other papers. However, in the early sentences that define $N$ and $m$ (i.e., "Suppose, the length of input data is ..."), none of the models succeed at predicting the correct notation for their first use, most likely because $N$ and $m$ never appear in the previous context.

This example also shows what appears to be a case of a model learning a conventional use of notation. Even the original BERT model is able to predict the correct usage of $N$ after the context defines the length as $N$ several times.

Analysis on the performance according to paper

| PaperID in S2ORC dataset: 16122894, ArXivID: 1408.3083, Section: Computational Complexity of Binarization Scheme | |
|---|---|
| `BERT:` | .. Suppose, the length of input data is , (Gold: $N$), , (Gold: $m$) is the number of source symbols, and , (Gold: $Y$) is the source. For the first symbol, the length of the binary string would be $M$ (Gold: $N$). The length of binary string for the second symbol would be the length of all the symbols, except the first symbol ( see Table 1 ). Likewise, the length of $n$ (Gold: $N$) binary string would be the length all symbols yet to be binarized. Mathematically, the length can be written as $N$, where $m$ is the probability of $m$ (Gold: $Y$) symbol. The total number of binary assignment would be $N$... |
| `BERT(FT):` | ... Suppose, the length of input data is $m$ (Gold: $N$), $m$ is the number of source symbols, and $n$ (Gold: $Y$) is the source. For the first symbol, the length of the binary string would be $N$. The length of binary string for the second symbol would be the length of all the symbols, except the first symbol ( see Table 1 ). Likewise, the length of $N$ binary string would be the length all symbols yet to be binarized. Mathematically, the length can be written as $N$, where $N$ (Gold: $m$) is the probability of $m$ (Gold: $Y$) symbol. The total number of binary assignment would be $N$ ... |
| `MathPred(FT):` | ... Suppose, the length of input data is $m$ (Gold: $N$), $n$ (Gold: $m$) is the number of source symbols, and $m$ (Gold: $Y$) is the source. For the first symbol, the length of the binary string would be $N$. The length of binary string for the second symbol would be the length of all the symbols, except the first symbol ( see Table 1 ). Likewise, the length of $N$ binary string would be the length all symbols yet to be binarized. Mathematically, the length can be written as $N$, where $m$ is the probability of $Y$ symbol. The total number of binary assignment would be $N$... |

**Table 7:** Example of paper-level predictions by MATHPREDICTOR and other baselines. We sequentially auto-suggest notations (left-only context) and concatenate the results. The notation tokens with gray background are the target. Blue colored notation tokens mean correct predictions and red colored notation tokens mean incorrect predictions. The gold labels (tokens) for the incorrect predictions are shown in parentheses.

domains, an example of mathematical operations are presented in Appendix E and F.

## 5 Discussion

**MATHPREDICTOR as an application.** Our proposed method showed reasonable performance in top-5 accuracy (71.3% and 80.0% for each task), which is strong for a novel task, but most likely not good enough for a real-world application.

However, when we sub-sample 10 times more than the main experiment, the performance on the tasks is improved on the same test set by +10% accuracy: 70.9% for top-1 accuracy and 81.6% for top-5 accuracy on auto-suggestion and 83.5% for top-1 accuracy and 89.0% for top-5 accuracy on consistency checks. These results are promising for MATHPREDICTOR to be utilized as an application.

**The results suggest that current models memorize the meanings rather than generalize over them.** Although we showed the possibility of modeling mathematical notations, most of the results indicates that the models are not able to predict tokens which have not been presented before (in Table 4). Predicting notation is a challenging problem, and relies on common patterns of notation usage across papers.

**Guidance for future work:** One way to enhance MATHPREDICTOR is to utilize the structure of notation. We attempted to encode the structure using token permutation (§3.3) but the method was not expressive enough. Future work could combine MATHPREDICTOR with direct modeling of mathematical notation using tree structures (Rabe et al., 2020). Sophisticated model structure to encode global context could bring further improvement.

## 6 Conclusion

In this paper, we propose two novel notation prediction tasks to evaluate mathematical notation semantics in academic paper writing. We then present a fine-tuned BERT particularly optimized on these tasks, which outperforms other baselines. Our analysis shows that the model can be thought of as a way to encode knowledge about the usages of mathematical notations in specific domains, although it does not seem to generalize beyond the notation within the text it is exposed to. We see this as a first step toward more powerful analysis tools that can one day act as a method to help authors of mathematical texts select and refine their notation.

## Acknowledgements

This research was supported in part by funding from the Alfred P. Sloan Foundation.

## References

Takeshi Abekawa and Akiko Aizawa. 2016. Sidenoter: scholarly paper browsing system based on pdf restructuring and text annotation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 136–140.

Akiko Aizawa, Michael Kohlhase, Iadh Ounis, and Moritz Schubotz. 2014. Ntcir-11 math-2 task overview. In *NTCIR*, volume 11, pages 88–98.

Maria Alexeeva, Rebecca Sharp, Marco A. Valenzuela-Escárcega, Jennifer Kadowaki, Adarsh Pyarelal, and Clayton Morrison. 2020. MathAlign: Linking formula identifiers to their contextual natural language descriptions. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 2204–2212, Marseille, France. European Language Resources Association.

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving bert a calculator: Finding operations and arguments with reading comprehension. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5949–5954.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. Scibert: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3606–3611.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Muthu Kumar Chandrasekaran, Anita de Waard, Guy Feigenblat, Dayne Freitag, Tirthankar Ghosal, Eduard Hovy, Petr Knoth, David Konopnicki, Philipp Mayr, Robert M. Patton, and Michal Shmueli-Scheuer, editors. 2020. *Proceedings of the First Workshop on Scholarly Document Processing*. Association for Computational Linguistics, Online.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

André Greiner-Petter, Abdou Youssef, Terry Ruas, Bruce R Miller, Moritz Schubotz, Akiko Aizawa, and Bela Gipp. 2020. Math-word embedding in math search and semantic extraction. *Scientometrics*, 125(3):3017–3046.

Andrew Head, Kyle Lo, Dongyeop Kang, Raymond Fok, Sam Skjonsberg, Daniel S. Weld, and Marti A. Hearst. 2020. Augmenting scientific papers with just-in-time, position-sensitive definitions of terms and symbols. *ArXiv*, abs/2009.14237.

Dongyeop Kang, Andrew Head, Risham Sidhu, Kyle Lo, Daniel S. Weld, and Marti A. Hearst. 2020. Document-level definition detection in scholarly documents: Existing models, error analyses, and future directions. In *SDP*.

Andrea Kohlhase, Michael Kohlhase, and Taweechai Ouypornkochagorn. 2018. Discourse phenomena in mathematical documents. In *International Conference on Intelligent Computer Mathematics*, pages 147–163. Springer.

Giovanni Yoko Kristianto, Akiko Aizawa, et al. 2014. Extracting textual descriptions of mathematical expressions in scientific papers. *D-Lib Magazine*, 20(11):9.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. 2020. S2ORC: The semantic scholar open research corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4969–4983, Online. Association for Computational Linguistics.

Sreekanth Madisetty, Kaushal Kumar Maurya, Akiko Aizawa, and Maunendra Sankar Desarkar. 2020. A neural approach for detecting inline mathematical expressions from scientific documents. *Expert Systems*.

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

Shuai Peng, Ke Yuan, Liangcai Gao, and Zhi Tang. 2021. Mathbert: A pre-trained model for mathematical formula understanding. *arXiv e-prints*, pages arXiv–2105.

Markus Norman Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. 2020. Mathematical reasoning via self-supervised skip-tree training. In *International Conference on Learning Representations*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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*.

EM Voorhees. 1999. Proceedings of the 8th text retrieval conference. *TREC-8 Question Answering Track Report*, pages 77–82.

Jevin West and Jason Portenoy. 2016. Delineating fields using mathematical jargon. In *Proceedings of the Joint Workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL)*, pages 63–71.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Magdalena Wolska and Mihai Grigore. 2010. Symbol declarations in mathematical writing.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Michihiro Yasunaga and John D Lafferty. 2019. Topiceq: A joint topic and mathematical equation model for scientific texts. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7394–7401.

Abdou Youssef and Bruce R Miller. 2018. Deep learning for math knowledge processing. In *International Conference on Intelligent Computer Mathematics*, pages 271–286. Springer.

Richard Zanibbi and Dorothea Blostein. 2012. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357.

## A Details on Manual Work

Specifically, we first filtered the notations using regex. When finding Numbers, we use ’([̂0̇-9; ]+? [a-zA-Z]*$)’ to find the numbers like “1cm”, “8K”, etc. However, LaTeX uses backslashes when describing both operations (e.g. “\times”) and symbols (e.g. “\alpha”). So we manually check whether their types are correctly classified. Finally, Letters are found by a regex ’(̂?!\)[a-zA-Z,]+?’. The rest are classified as math operations and symbols, after another manual investigation. Since the number of unique notation tokens is relatively small in our test set (e.g.,Letter:169, Number:56, OpSymbol:108, LatexMacro:34), this manual procedure is highly accurate.

## B Effect of Masking Length Constraint

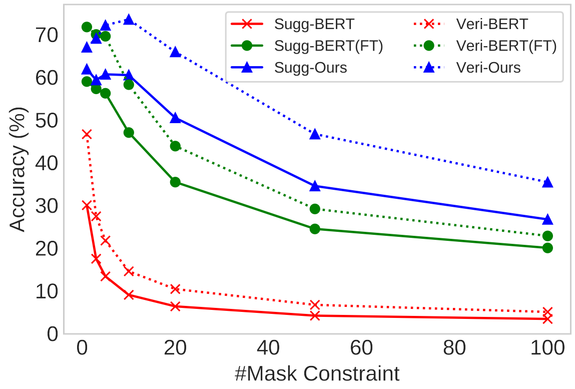


**Figure 5:** The performance according to the change in the number of mask constraint.

During the preprocessing step, we skipped sentences which had more than 10 tokens of mathematical notation. Here, we change the masking length constraint from 1 to 100 and report the results in Figure 5. These results show that increasing the number of masks to more than 10 largely degrades the performance.

## C Model Training Details

**Baseline training details.** In the original BERT’s masked language model training, it first selects a part of tokens (in default 15%). Among them, 80% tokens are masked, 10% tokens are randomly changed to other tokens, and 10% remains original. Only for the masked tokens, the model calculates the loss between the predicted tokens and the gold tokens.

The training batch size is 6, using adam with $\beta_1$ of 0.9, $\beta_2$ of 0.999, and $\varepsilon$ of 1e-8. The number of maximum training epoch is 20 but the training stops about 15 epochs by early-stopping. We stop the training if the validation accuracy does not increase in 3 epochs.

**Hyperparameters.** We use the default hyperparameters of hugging face (Wolf et al., 2020) except for learning rate. It uses gelu function as activation functions, and dropout with 0.1% probability. Layer normalization is also applied with $\varepsilon$ of 1e-12. The hidden size is 768 with attention heads of 12, and the number of hidden layers are 12. all the weights are initialized by 0 mean and 0.02 standard deviation. The maximum input token length, which is the same with position embedding length is 512. The token vocabulary size depends on the predefined vocab but 28,996 in BERT cased version, 31,116 in SciBERT cased, and 50,265 in RoBERTa. Note that we added custom vocabulary such as LaTeX macros.

**Computing resources.** We train the data using 2 Titan Xp (12G) GPUs. In our dataset, it takes 2 hours per epochs with 6 batch size. Although we collected larger version of dataset, training over 2M sentences (10,000 papers X 234 sentences per paper) takes too much time (more than 24 hours per 1 epoch using our computing), so we mainly report experiment results with the smaller version.

## D Effect of Context Size

Depending on the local context size, MathPredictor’s performance varies. We performed an ablation study on performance changes according to the number of local context sentences in training and testing. Table 8 and Table 9 show the notation auto-suggestion performance and the consistency check performance, respectively. We observe that using the same size of local context for both training and testing shows the best result. Also for notation prediction task, using seven left sentences for both training ($L_7$) and testing ($L_7$) shows the best performance even better than using the maximum token length of BERT ($T_{512}$). For notation consistency task, there was some variation but using three left and three right sentences for both training ($L_3R_3$) and testing ($L_3R_3$) gives the comparable performance. We use the best setting ($L_7 - L_7$ for suggestion and $L_3R_3 - L_3R_3$ for consistency check) in the aforementioned experiments.

| | | #Context Sentence | | | | |
|---|---|---|---|---|---|---|
| | | $L_1$ | $L_3$ | $L_5$ | $L_7$ | $L_{MAX=T_{512}}$ |
| #Train. | $L_1$ | **50.32** | 50.53 | 48.84 | 46.28 | 39.18 |
| | $L_3$ | 47.93 | 58.23 | 61.12 | 56.85 | 53.96 |
| | $L_5$ | 48.61 | 58.35 | 61.90 | 59.53 | 57.98 |
| | $L_7$ | 48.82 | **60.51** | **62.64** | **60.54** | 60.67 |
| | $L_{MAX}$ | 47.56 | 55.78 | 57.08 | 59.53 | **63.07** |

**Table 8:** Top-1 accuracy on the notation prediction. $L_n$ denotes left $n$ sentence(s) and $T_{512}$ denotes left 512 tokens, which means the maximum size of BERT. We report various training (encoding) and evaluation (decoding) settings that fine-tunes our model using $\{L_1, L_3, L_5, L_7, T_{512}\}$ and predicts the symbols with $\{L_1, L_3, L_5, L_7, T_{512}\}$ context.

| | | #Context Sentence | | |
|---|---|---|---|---|
| | | $L_1R_1$ | $L_2R_2$ | $L_3R_3$ |
| #Train. | $L_1R_1$ | **66.65** | 70.35 | 69.10 |
| | $L_2R_2$ | 64.86 | 70.43 | 72.77 |
| | $L_3R_3$ | 66.31 | **72.33** | **73.54** |

**Table 9:** Top-1 accuracy on the notation consistency task. $L_nR_n$ denotes left $n$ and right $n$ sentence(s). We report various training (encoding) and evaluation (decoding) settings that fine-tunes our model using $\{L_1R_1, L_2R_2, L_3R_3\}$ and predicts the symbols with $\{L_1R_1, L_2R_2, L_3R_3\}$ context.

## E Performance by Domain

Table 10 shows the performance according to paper domains. Among the test data, we select four domains according to the number of data: Computer Science (CS), Mathematics (Math), Physics, and Statistics (Stat). The result shows that Physics shows the best performance and Statistics are the worst, while the others are similar performance to overall performance. It might be the use of notations in Physics are relatively short and simple with strict notation rules. Meanwhile, in Statistics there are a series of numbers, which is hard to predict by our method, as seen in the previous examples.

## F Semantics behind Mathematical Operations

.

We also investigate whether the models can predict notations for mathematical operations. We first select examples from the challenge set to exclude simple memorable symbol patterns from context, and extract test samples that contain mathematical operations. Then we intentionally mask individual tokens and check whether the model can successfully predict them from the rest.

| | Suggestion | | | Consistency | | |
|---|---|---|---|---|---|---|
| | Top1 | Top5 | MRR. | Top1 | Top5 | MRR. |
| CS | 60.34 | 71.09 | .655 | 73.98 | 80.41 | .772 |
| Math. | 61.93 | 73.53 | .673 | 70.23 | 77.28 | .737 |
| Physics | 80.30 | 84.85 | .826 | 83.33 | 84.85 | .838 |
| Stat. | 52.71 | 65.12 | .584 | 65.80 | 74.46 | .700 |

**Table 10:** The comparison of domain-level top-1 accuracies and mean reciprocal rank scores in both tasks.

Table 11 shows one of the examples. We mask each token in the symbol notation '4 * 4 = 16' and compare models' outputs. our method shows that it fails to (auto-)suggest but succeeds to verify the numbers or operators. We believe that the model can be much improved by training our model on much larger dataset like entire S2ORC dataset.

## G Additional Examples

$\cdots$ Within 5% visual Complete difference, we subgroup them based on 4 conditions of SI difference : $SI_{diff} >= 10$ ; $1 <= SI_{diff} < 10$ ; $-10 < SI_{diff} <= -1$ ; $SI_{diff} <= -10$. Within each SI difference condition, we again subgroup each of them into 4 conditions of PSI difference: $PSI_{diff} >= 10$ ; $1 <= PSI_{diff} < 10$ ; $-10 < PSI_{diff} <= -1$ ; $PSI_{diff} <= -10$. In total, we have $4*4=16$ conditions for our experiment. The reason we selected our video pairs based on SI and PSI for Phase-1 experiment is that we believed these are key QoE metrics to best express user perception. $\cdots$.

| | Masking | [MASK] $*4=16$ | 4[MASK]$4=16$ | $4*$[MASK] $=16$ | $4*4$[MASK]16 | $4*4=$ [MASK] |
|---|---|---|---|---|---|---|
| Suggest. | BERT | +, conditions, - | +, '.', / | =, $, 1 | *, +, / | 4, 5, 6 |
| | BERT(FT) | 2, **4**, 10 | $\psi$, /, + | :, *, $\psi$ | *, /, $\psi$ | 5, 4, 10 |
| | MathPredictor | {, \, v | {, p, v | {, p, 2 | **=**, _ , $ | 5, $, 6 |
| Consist. | BERT | 1, 2, + | +, -, / | <, +, = | *, +, / | 10, 20, 11 |
| | BERT(FT) | 2, **4**, 1 | ',', =, + | *, t, 10 | **=**, \mathclose, + | 10, 5, 4 |
| | MathPredictor | N, 10, K | =, '.', ^ | **4**, 10, 2 | **=**, ^ , _ | 10, **16**, 4 |

**Table 11:** Examples of top-3 predictions for mathematical operators from the challenge set. We partially mask the notation tokens $4*4=16$ by masking each token. The blue means correct notation predictions, matching the gold ones.

.. Note that in this case $n+4$ . As $n+4$ , we get that $n+3$ . Essentially the same calculation works if $n+4$ is close, from below, to a power of 2, as then $n+3$ is not much larger than $n$ .

.. The global optimality condition holds for communities $c$ and $c'$ when no other pair of communities could be merged so as to increase the modularity more than would merging $c$ and $c'$. The local optimality condition weakens the global condition, holding when no pair of communities, one of which is either $c$ or $c$ *preliminary* (Gold: $/$), could be merged to increase the modularity more than would merging $c$ and $c$ *preliminary* . (Gold: $/$)

.. Thus, under appropriate technical conditions, the chain has a unique stationary distribution and the sequence converges in distribution to this invariant distribution. Let *Watts* (Gold: $\Upsilon$) $_n$ denote a right - invariant distance on the group $K_n$ .

The main benefit of the filter is with respect to its convergence properties. Indeed, under very mild conditions, the covariance matrix $L$ (Gold: $P$) $_n$ and the filter's gain $L$ (Gold: $K$) $_n$ are proved to converge to fixed values CITATION.

As $C=G$ is compact ( as a closed subset of a compact ) the open cover $K=\{x\in G, h(x,0)=h(I_d,0)\}$ has a finite subcover and there exists an integer $K$ such that $G$, i. e. $C=G$ .

The set of all strategies is denoted by $\mathsf{N}\in N^*$ The set $B$ denoting $\{0,1\}$, let $f:B^{\mathsf{N}}\longrightarrow B^{\mathsf{N}}$ be a function and $S\in\mathbb{S}$ be a strategy. The so - called chaotic iterations are defined by $B$ and $lit$ (Gold: $\{$) $0,1\}$

In addition, $\lfloor d(X,Y)\rfloor$ is a measure of the differences between strategies $n$ and $E,\check{E}$. More precisely, this floating part is less than $\lfloor d(X,Y)\rfloor$ if and only if the first $n$ terms of the two strategies are equal. Moreover, if the $\lfloor d(X,Y)$ $)$ (Gold: $\rfloor$) digit is nonzero, then the $n$ terms of the two strategies are different.

i. e., the number of distinct encoded rows stored across the $\{_t :\in \{\mathscr{C}_{\mathscr{K}} : |\mathscr{K}\cap\mathscr{Q}\backslash\{k\}| \neq 0\}, t\in\mathscr{R}_k\}$ servers is exactly $\mathscr{Q}\backslash\{k\}$. As a result, the communication is finished with the load $s_q = s_{min}$ for this case. Case 2 : $s_q$ $=$ (Gold: $<$) $s_{min}$ .

**Table 12:** Example of predictions by our model in test data. The notation tokens with gray background are the target. Blue colored notation tokens mean correct predictions and red colored notation tokens mean incorrect predictions.