,,

Universiteit Leiden
The Netherlands

Bachelor Datascience and Artificial Intelligence

Improving the computational efficiency of handwritten mathematical expression recognition

Luis Kleinwort

Supervisors:
Dr. Hazel Doughty
MSc. Luc Sträter

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/07/2025

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr. Hazel Doughty, for giving suggestions for improvement for several methods and providing helpful feedback for the writing of the thesis throughout the semester.

This work was performed using the compute resources from the Academic Leiden Interdisciplinary Cluster Environment (ALICE) provided by Leiden University.

Abstract

Handwritten mathematical expression recognition (HMER) is a subfield of optical character recognition (OCR) that has much potential for easier interaction with mathematical applications especially in educational settings. This thesis focuses on making a cost-effective, real-time capable HMER model by reducing the computational cost of a TrOCR-based model while maintaining a similar prediction accuracy. Several methods are explored: a latex-specific tokenizer is introduced, a tree-based inference approach is tested and finally the model is distilled. The models are mainly trained and tested on the MathWriting [GFM24] dataset. The final model achieves an accuracy of 0.79% and requires only around 34 GFLOPs per prediction. For some training runs the ALICE GPU cluster of Universiteit Leiden was used.

Demo

A demo of the model can be found at `https://lk-bachelorproject.fly.dev/` The username and password is bachelorproject, demo_HMER. After the submission of this thesis, this demo will be maintained on a best-effort basis. The used model might not reflect the final model of this thesis.

Contents

1 Introduction

Handwritten mathematical expression recognition (HMER) is a specialization of optical character recognition (OCR). HMER could allow for easier interaction with digital math applications, especially in educational settings. However, except for Photomath[Pho], HMER is not widely used as a way to input mathematical expressions in applications. This might be due to the fact that HMER is a much more difficult task than traditional OCR. The main reason for this is that no large high-quality datasets exist that fully represent all of the various types of mathematical expressions. This variety arises from the fact that mathematical expressions are used in many fields, such as linear algebra, calculus, physics discrete math, logic et cetera. HMER can also not easily make use of recent advances in OCR methods, since mathematical expressions do not have a linear order like normal text and instead require a tree-representation.

In this thesis I will focus on improving the computational cost of running HMER inferences with a TrOCR-based model in the setting of digital handwritten mathematical expressions input with the goal of making an HMER model that is suitable for use in a "math expression input field" for computer applications. The main priorities for a such a model are a very low computational cost, high accuracy and low bias towards commonly used mathematical expressions. Having a low bias is important mostly for educational applications, where students might be asked to remember and write out some formula. In this scenario, a unclear and slightly wrong variant of a well known formula, should not be "autocorrected" to the correct version since this could be abused by students for cheating.

The research question of this thesis is the following: How can the computational efficiency of a model for handwritten mathematical expression recognition be improved without compromising accuracy?

2 Related Work

The OCR task has been extensively researched for several decades. This has lead to a wide availability of OCR models that can be used for real-time OCR even on devices with limited computational capabilities such as phones. Examples are STRIDE [MPA+21], Context-Free TextSpotter [YTD+21] and PP-OCRv3 [LLG+22]. In contrast to this, the HMER task has been researched significantly less. This might be due to the HMER task being inherently more difficult: Unlike general OCR where the characters usually follow a simple left-to-right order, HMER has the additional complexity of having to model the relationship between characters based on their position relative to each other [And67]. HMER has also likely been researched less because its applications are not as broad as OCR in general.

Previous approaches in the HMER field can be categorized into end-to-end models and multi-step pipelines that usually first detect characters and then use some heuristics or another model to infer a latex expression from the detected characters. An example of a multi-step approach is the method proposed by Le and Nakagawa [LN16]. Most end-to-end models use an encoder-decoder model similar to the non-HMER TrOCR model that is used as a basis for the models in this thesis. The "Bidirectionally trained transformer" (BTTR)[ZGY+21] expands on the usual left-to-right decoding approach by adding a right-to-left variant of the expression, which is decoded simultaneously in order to improve prediction robustness. PosFormer[GLSY24] has a much more

conventional encoder-decoder approach, but it adds positional information during the training phase in order to achieve better model performance. This extra information is omitted during the inference phase.

2.1 TrOCR

In order to answer the research question, a model that has not yet been optimized for computational efficiency with decent accuracy is required. The previously mentioned HMER models are not a good starting point since they have already been optimized for computational efficiency. Therefore, the non-HMER TrOCR [LLC+23] model has been chosen as the base model for this thesis. The TrOCR model is a powerful vision encoder-decoder model with a focus on making use of well-pretrained encoder and decoder models. One advantage of this model is that it is an end-to-end model, meaning that only minimal pre- and postprocessing is required for an inference and retraining it for the HMER task does not require any additional logic. Additionally, apart from not being a HMER model, its single line OCR task is similar to the goal of this thesis, as it is also focused on single-line HMER. Lastly, it has a variant that is finetuned for handwritten lines of text. This makes it an ideal starting point for this thesis.

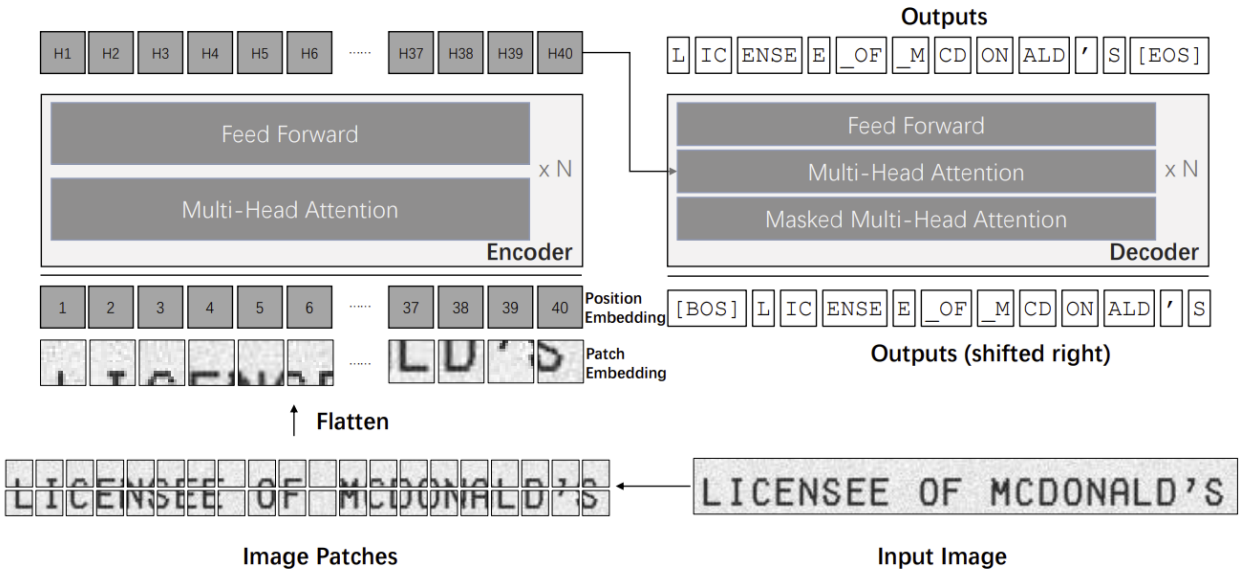


Figure 1: An overview of the TrOCR architecture[LLC+23]

As seen in Figure 1 the inference is made up of several steps: First, the picture is split up into image patches that are flattened and embedded using a patch embedding and a position embedding[LLC+23]. These patches are then passed as tokens to the transformer vision encoder, which encodes the information of the input picture in a fixed-size latent space. The decoder uses the encoder outputs to auto-regressively predict the expression in the picture until it predicts the end-of-sentence token. The last step is to use the tokenizer to detokenize the predicted tokens into a string.

The reason for the strong performance of the TrOCR model lies in the several training stages[LLC+23]: In the first pretraining stage, the model is trained on 684 million synthetic image and label pairs. In the second pretraining stage the model is trained on a different, smaller synthetic dataset. The final training stage is the finetuning stage, where the model is trained on a human-labeled dataset that is specific either to handwritten or printed text.

It is important to note that the different variations of the TrOCR models (base, small) differ in more aspects than just number of layers: They use different base-models for encoder, decoder and tokenizers.

The base model requires around one second of computation time on a modern multicore processor, which makes realtime text-input on lower end processors found in typical phones infeasible. In this thesis, most attention is put on trying to improve the computational requirements of the decoder.

3 Methods

3.1 Problem definition

The single-line HMER task that this thesis addresses can be described as follows: The input is an image $I \in \mathbb{R}^{H \times W \times 3}$ of some mathematical expression with arbitrary dimensions H and W with white background and black text. The expression should be roughly centered in the image and it should be legible without too much difficulty. In the digital HMER context, the input image I can also be represented as a set of line-traces instead of raw pixel values. The output should be the math-mode latex-representation of the shown expression, excluding the dollar-sign math-environment delimiters that are used in latex.

3.2 TrOCR preprocessing

Since the TrOCR model requires that the input image has some fixed, square dimensions, the image first needs to be preprocessed. This is done by the TrOCR processor, which stretches the image to the expected size. Additionally, the color of the image is normalized.

During training, the labels also have to be preprocessed by using the models tokenizer. The tokenizer takes a the string of some label as the input and outputs a one-hot encoded array $L \in \mathbb{R}^{N \times |V|}$ where N is the number of tokens that are required to encode the string and $|V|$ is size of the vocabulary of the tokenizer. $|V|$ matches the amount of output that the decoder has at each decoding step. To allow for batched training, the tokenized arrays are padded with a predefined padding token at the end so that they have matching dimensions and can be represented as a single tokenized label tensor.

3.3 Rendering

The final picture that is given as an input to the model should also be invariant to the resolution of the screen that the user was writing on (scale invariance) and it should also always be positioned in the center of the input image.

Digital HME are usually not represented as the image that the user saw when writing the expression to the screen. Instead, digital HME are usually represented as 2D-point traces, so they

first need to be rendered to an image before they can be given to the Vision-Encoder. It is important that the traces are rendered the same way during training as during inference to minimize the domain-gap between the training and the target domains. Using the original picture that was shown to the user is inpractical because every device and application might render the expression in a slightly different resolution and style and trace-data is a much more compact representation. In order to achieve the required centering and scale invariance, the trace points are first centered around their mean and then scaled by the reciprocal of the variance of the distances of all points to the center. This normalizes the point positions without distorting the width and height dimensions.

3.4 Training

The model training process follows the same conventional image-to-text as the final training stage used by TrOCR. In each iteration, the input images are rendered, stretched and normalized to fit the input size of the encoder. The labels are tokenized, one hot encoded and padded, so that all labels of the batch have the same length. The image is passed to the network and the decoder predicts logits for all output token positions simultaneously. Since the model sequentially predicts tokens during inference, an attention mask that prevents tokens from attending to future tokens is used in the training process. After the model computes the logits, token probabilities are calculated using softmax and the cross-entropy loss between the tokenized labels and the predicted token probabilities is computed. Finally, the loss-gradient is calculated and the models weights are updated using the AdamW optimizer [LH19]. The hyperparameters that are used are specified in Appendix 8. The training implementation of this thesis is based on the implementation of Romeo Sommerfeld [Rom] which itself uses the Huggingface Transformers library[hug] and the PyTorch library [PyT].

3.5 Data Augmentation

Data augmentation is used to decrease overfitting. Specifically either random rotation of up to 15 degrees, random gaussian blur or randomly zooming out by up to 30 percent is applied.

3.6 Custom tokenizer

The tokenizer of the TrOCR model was originally trained for English text, which results in a vocabulary that contains many tokens that are complete or parts of English words. These tokens are not useful in the context of HMER, as mathematical expressions rarely contain English words. On top of this, most of these English word tokens do not occur in the training datasets, which means that the model would not use them even if an expression with an English word was given to it.

The main drawback of having a large vocabulary is that the projection layers of the decoder model have unnecessarily many parameters, which increases the RAM footprint of the model significantly, especially during training.

To address this, the first method in this thesis is to change the tokenizer vocabulary to be a better fit for the HMER task. The MathWriting Dataset [GFM24] contains a complete list of all tokens that make up its labels, which will be used as the vocabulary for the new custom tokenizer Math Tokenizer 1 (MT1). These tokens include such as upper- and lowercase letters from the Latin

alphabet, LaTeX symbols for Greek letters like π and LaTeX macros like `\frac`. The full tokenizer vocabulary derived from the MathWriting Dataset is listed in Appendix 8

One drawback of using the MathWriting tokens as the vocabulary is that they do not include tokens for common mathematical functions like `sin` and `log`. As a result, the tokenized representation of many expressions will be longer, requiring more computation time for decoding. This could easily be resolved by adding character sequences that are common in mathematical expressions as tokens.

Using LaTeX tokens also has the advantage that LaTeX macros such as

```
\sqrt
```

can be represented as a single token instead of having to be split up into

```
\ sq rt
```

This has two effects: The decoding time of latex macros is reduced, since fewer autoregressive prediction steps are required for them. Secondly, the decoder model does not have to learn which token combination refers to which LaTeX macro. However, it is likely that this effect only becomes apparent when the decoder model does not have extra capacity to learn such patterns, which seems to only be the case for the distilled version of the decoder.

```
original:   27   \ frac { V _{ i } }{ T _{ i }} = \ frac { V _{ f } }{ T _{ f }}
adjusted:   33   \frac { V _ { i } } { T _ { i } } = \frac { V _ { f } } { T _ { f } }

original:   15   \ til de { 0 _{ P }} / 0 _{ P }
adjusted:   16   \tilde { 0 _ { P } } / 0 _ { P }

original:   13   \ int \ frac { 1 }{ x } d x
adjusted:   12   \int \frac { 1 } { x } d x
```

Figure 2: Tokenization comparison of original trocr-small tokenizer vs Custom tokenizer MT1

In Figure 2 some examples of tokenized expressions are shown. It shows how the token boundaries are better aligned with the LaTeX tokens, but it also shows the disadvantage of using this LaTeX-based tokenizer: some expressions require more tokens overall.

In order to address this, two more tokenizers will be tested: The MT1 tokenizer will be extended by adding tokens for common mathematical functions such as “sin” and “cos”. The resulting tokenizer is called MT4 in the experiments. Lastly, another tokenizer MT5 will be extended with common syntactical combinations such as a double closing and opening bracket. Additionally, tokens that are always followed by an opening bracket will be modified to contain the opening bracket themselves. With these tokenizer variants it can be measured if there is a tradeoff between speed and token information density.

3.7 Decoder Distillation

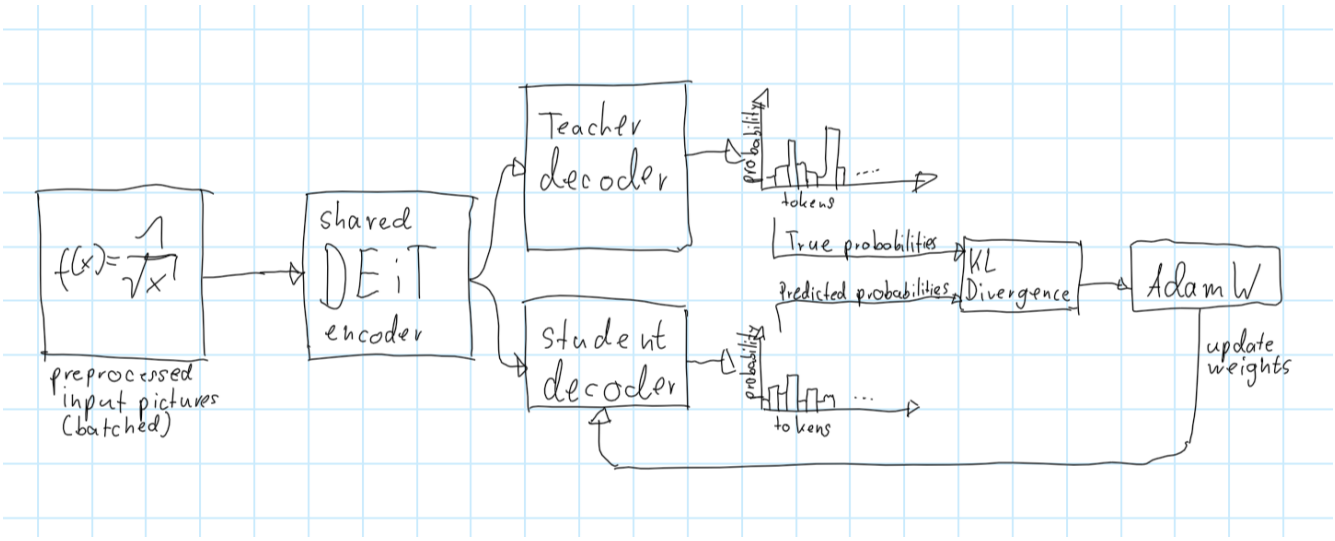


Figure 3: Illustration of the decoder distillation process

A widely used technique of making models smaller and more computationally efficient to run is knowledge distillation. In knowledge distillation, a smaller "student" model is created and trained to mimic the outputs of the original "teacher" model. There are several different ways of defining the distillation loss function, such as defining a loss that makes the students internal transformer layers output similar values as the corresponding transformer layers in the teacher. Another way of increasing the similarity between the student and teacher models is to add similarity based loss between the embedding layers of the student and teacher model. However, in this thesis a simpler distillation loss is used, that only trains the student model based on the similarity of its outputs and the teacher models outputs. For this, the Kullback–Leibler divergence between softmax-temperature-scaled output probabilities of the student and teacher models are calculated. This distillation method allows the student model to have a completely different internal architecture than the teacher model. This method has first been introduced by Hinton et al[HVD15]. The student models are initialized using the teacher models' encoder and a randomly initialized decoder with a reduced number of transformer layers. During distillation, the only parameters that are trained are the students' decoder parameters in order to improve training and convergence speed. This process is illustrated in Figure 3.

3.8 Tree-based inference approach

The tree-based inference approach is an alternative to the usual linear decoding method used by most text-decoder models. The motivation for a different approach is that predicting latex expressions using this method allows for preventable syntactical latex errors, such as not providing the correct amount of arguments to a macro or having mismatching curly-brackets. In order to avoid these syntactical errors the decoder needs to keep track of syntactical features of the latex expressions like number of opened curly-brackets, which argument of which latex macro it is currently decoding, etc. Consider the case shown in Figure 4, 5: The decoder needs to be able

to decide wether to predict the “i” that followed the first “T” or to predict the “f” that is the subscript of the other “T”. Additionally, it needs to remember to close the curly-bracket of the subscript and the fraction macro.

In order to decrease the potential mistakes that the decoder can make when dealing with the syntax of the expression the following tree-based inference approach was implemented: When the decoder predicts a latex macro that has arguments such as `\frac` or the start of a sub/superscript using `_` or `*` the decoding process is split up into one branch for every argument of the macro and one branch for the part of the expression that follows the current macro. To indicate the current branch to the decoder, special branch indicator tokens such as `<FRAC1>`, `<FRAC2>` are inserted. The `<AFTER>` token is used to indicate the part of the expression that follows the current macro.

Note that the resulting tree representation is different from the expression trees that represent valid mathematical expressions. The tree representation in this thesis was made to still look similar to the latex representation and to allow for invalid mathematical expressions.

An example of the resulting tree is visualized in Figure 6. During training, each sample is copied for every possible tree path and the loss is disabled for the branch indicator tokens, since these are inserted using custom decoding logic during the inference phase. For the visualized the list of tree paths is the following:

```
\frac<FRAC1>V_<_>i
\frac<FRAC1>V_<AFTER>
\frac<FRAC2>T_<_>i
\frac<FRAC2>T_<AFTER>
\frac<AFTER>= \frac<FRAC1>V_<_>f
\frac<AFTER>= \frac<FRAC1>V_<AFTER>
\frac<AFTER>= \frac<FRAC2>T_<_>f
\frac<AFTER>= \frac<FRAC2>T_<AFTER>
\frac<AFTER>= \frac<AFTER>
```

This approach might increase the importance of earlier tokens during training which could be addressed by a loss-scaling mechanism, however this might be counteracted by the fact that earlier tokens also have a higher importance during inference; During training, the decoder is trained assuming that the previous tokens have been predicted correctly meaning that one early mistake will likely cause additional errors.

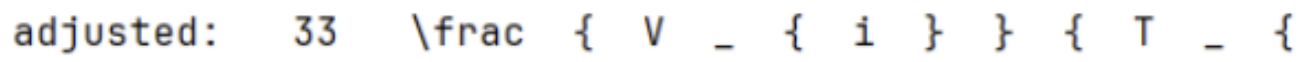


Figure 4: An example of an in-progress decoding

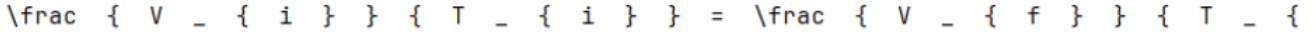


Figure 5: Another example of an in-progress decoding, paused at the second T _ subscript.

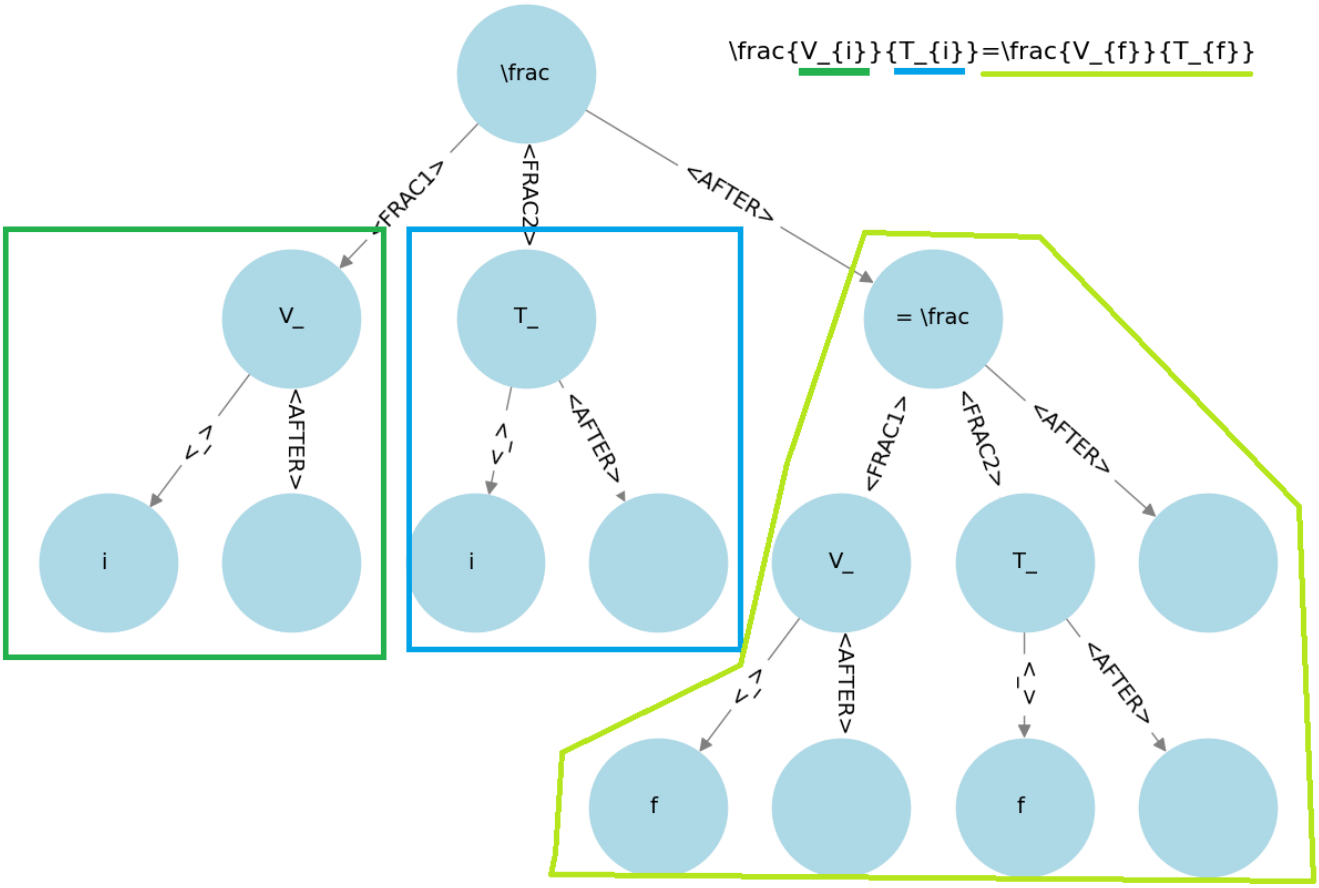


Figure 6: A visualization of the tree representation of $\frac{V_i}{T_i} = \frac{V_f}{T_f}$. The outlines show which subtree corresponds to which part of the expression.

```
\frac<FRAC2>T_<_>
\frac<AFTER>= \frac<FRAC2>T_<_>
```

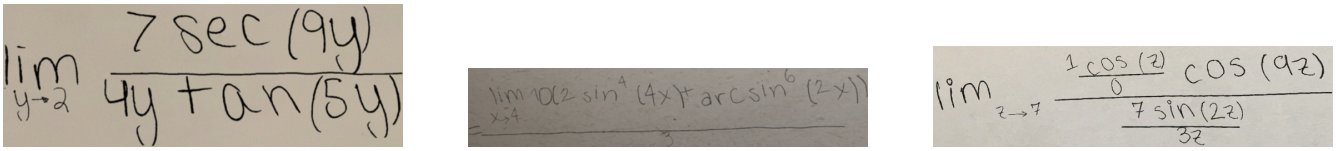
4 Datasets

In this thesis, I have been developing the model with several datasets. Each dataset is split into a 5% validation and a 95% training split. The validation split has been set to 5%, since inference iterations of this model are much slower than training iterations. Ideally a larger test split percentage should be used, however since the main dataset that the models are tested on is already quite large 5% is likely still sufficient to draw conclusions.

4.1 Aida Calculus Math Handwriting Recognition Dataset

The Aida dataset is a dataset of 100_000 synthetic images of HME, which have the appearance of photos of HME written on paper [Pea]. The disadvantage of this dataset is that the structure of the expressions lacks variety, as most of the expressions in this dataset start with *lim*.

The Aida dataset is available on kaggle, uploaded by "Aida by Pearson". At the time of writing, there seems to be no papers associated with this dataset.



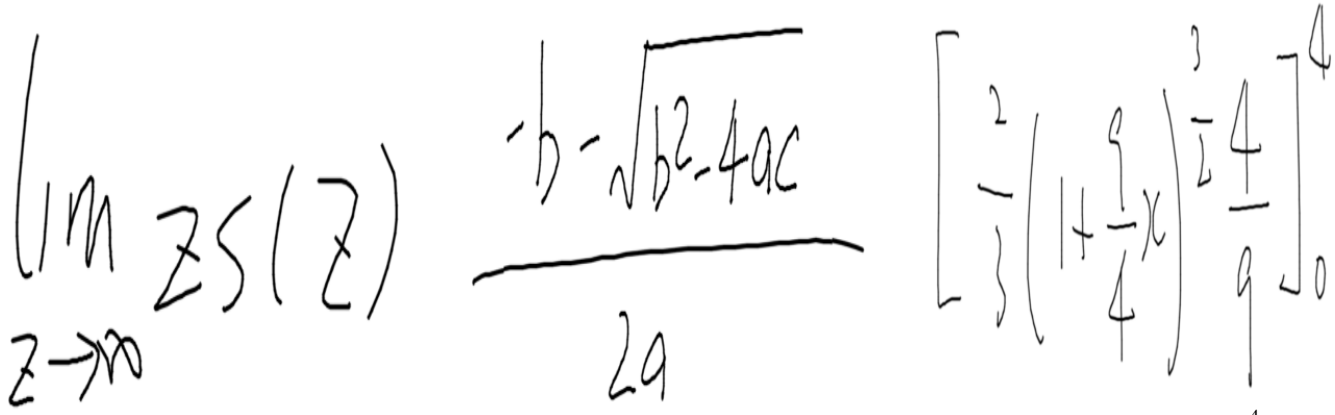
(a) (b) (c)

Figure 7: Three examples of the Aida dataset

4.2 Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME)

The CROHME 2023 is the 7th competition of its series. Each competition is associated with a dataset of digital HME, which was expanded regularly. The CROHME dataset is the most commonly used dataset in the digital HME research, since until recently very few other datasets of digital HME were available publicly. The CROHME datasets each have a dedicated test split.

[XMSL+23]



(a) $lim_{z \to \infty} zs\,(z)$ (b) $\frac{-b-\sqrt{b^2-4ac}}{2a}$ (c) $\left[\frac{2}{3}\left(1+\frac{9}{4}x\right)^{\frac{3}{2}}\frac{4}{9}\right]_0^4$

Figure 8: Three examples of the CROHME dataset as they are presented to the model after being stretched by the preprocessing step.

4.3 MathWriting (MW)

The MathWriting dataset is a large and high quality dataset containing 230_000 digitally written HME in its training split [GFM24]. However, since it has only recently been published, few research has been based on it. It also provides a validation data test split with 16_000 samples and a test split with 8_000 samples. Additionally, it has 396_000 synthetic samples which generated by combining characters of real samples. It is unknown how many participants have contributed, however it is stated that approximately 150 different device types have been used.

(a) $\frac{V_i}{T_i} = \frac{V_f}{T_f}$ (b) $\tilde{O_P}/O_P$ (c) $\int \frac{1}{x} dx$

Figure 9: Three examples of the MathWriting dataset as they are presented to the model after being stretched by the preprocessing step.

5 Metrics

In order to measure if and how much the described methods can improve the models computational efficiency without compromising on accuracy several metrics are required.

5.1 Accuracy

In this thesis, the accuracy is defined as the proportion of predictions where the final latex-expression strings are an exact match to the label given as a value between 0 and 1. There are three other definitions that could be considered: Exact match of the tokenized label and predicted tokens, exact match of the latex tokens or exact match of the syntax trees of the two expressions. Comparing the tokenized expressions could lead to under-reporting of the perceived accuracy, as one string can have multiple tokenized representations with some vocabularies. Comparing the syntax trees to calculate accuracy would be ideal, as this would measure the quality of the end result most precisely.

The disadvantage of the accuracy metric is that it does not distinguish between a completely wrong prediction and a slightly wrong prediction. Due to this the Character Error Rate metric is also employed.

5.2 Character Error Rate (CER)

A commonly used metric for OCR models is the CER metric proposed by Thennal et al. [KJGK24] for automatic speech recognition (ASR). It is an edit-distance based metric that is defined as follows:

$$CER = \frac{S + D + I}{N}$$

Where S is the number of character substitutions, D is the number of character deletions and I is the number of character insertions needed to modify the label into the predicted label. N is the

amount of characters in the label. As in the MathWriting[GFM24] paper, the MathWriting tokens are used instead of characters, since long latex macros would otherwise strongly skew the metric.

In order to make the results in this thesis comparable to other models that are tested against the MathWriting[GFM24] dataset test split.

5.3 GFLOPs

Since the goal of this thesis is to increase the computational efficiency of this model, it is important to have a metric that measures how high the computational requirements of the model are. One such metric is the Giga-Floating-Operations (GFLOPs) metric, which is simply the amount of floating-point operations needed for some inference. It is important to note that the amount of GFLOPs needed depends on the output itself for this model, since the decoder continues predicting more tokens until it outputs an end-of-sentence token (or the limit is reached). The tokenizer also has an effect on this, since it determines how many tokens are needed to represent a given expression.

To measure the GFLOPs, a pre-set expression is tokenized, a random input is given to the model, but at each decoding step the logits of the model are ignored and the pre-tokenized array of tokens is forced to be the output. The GFLOPs are measured using the PyTorch profiler API. This way, the real-world GFLOPs of the model are calculated with the assumption that the output is correct. However, GFLOPs do not provide a full picture of the computational requirements of a model, since there are more aspects such as floating point precision and the RAM requirement during inference. This process is repeated for 6 randomly chosen expressions of the MathWriting dataset:

```
p(z)=\prod_{n}(z-c_{n})
argmax_{W}\prod_{v\in V}P(v)
AI_{T}=100\times\frac{d}{n}
B=\frac{200+p}{200-p}
\frac{\frac{64}{252}}{(\frac{3}{\sqrt{10}})^{476}}
\hat{\alpha},\hat{\beta}
```

6 Experiments

In order to measure the impact that each of the previously described methods has, hyperparameter-tuning experiments are run for each method that introduces new hyperparameters in order to show the true potential of each method. In each series of experiments, the configuration that performed best in the previous set of experiments is used as the base configuration.

If not specified otherwise, the MathWriting dataset will be used to train the models.

6.1 TrOCR model variant experiments

To find out which TrOCR variant is best suited as the starting point for this task, the two TrOCR variants “microsoft/trocr-base-handwritten” [Mica] and “microsoft/trocr-small-handwritten” [Micb] are trained on the MathWriting dataset. The difference between the models is not only the number of parameters, but also the model architectures of the encoder and decoder models. The base

variant uses the BEiT vision encoder with a RoBERTa text decoder which results in it having 333 million parameters. The small variant uses the DEiT vision encoder and a UniLM text decoder which results in 66 million parameters. In these experiments the only previously described method that is applied is the rendering 3.3 method, in order to have a baseline that the following methods can be compared against. The results will be shown in Table 1

6.1.1 Results and Discussion

Table 1: Model Comparison Results

Method Name	MW Validation Accuracy (↑)	MW Validation CER (↓)	GFLOPs (↓)
Base TrOCR Variant	0.91	2.13	553.34
Small TrOCR Variant	0.87	2.50	58.56

From Table 1 it can be seen that the smaller TrOCR variant is a good starting point, since it only has a slightly smaller accuracy and requires almost 90% less GFLOPs than the base variant.

6.2 Dataset experiments

Another series of experiments will show how the datasets can complement each other: A model will first be trained on only MathWriting and once with all three mentioned datasets. To measure how the other datasets affect the models performance on the MathWriting dataset, the accuracy metrics will only be calculated using the MathWriting dataset in these experiments. The hypothesis is that adding more datasets could allow the model to generalize better and therefore also allow it to perform better on the MathWriting Validation set. In these experiments the small model variant with the original tokenizer will be used. The results will be shown in Table2.

6.2.1 Results and Discussion

Table 2: Dataset Comparison Results

Method Name	MW Validation Accuracy (↑)	MW Validation) CER (↓)
MW (previously small variant)	0.87	2.50
CROHME + AIDA + MW	0.76	4.47

As Table 2 shows, did not improve the accuracy on the MathWriting Validation set, proving the previously made hypothesis wrong. One issue that might prevent the model from being able to generalize better is that the latex expressions are not normalized consistently between the different datasets and follow slightly different latex conventions. This means that the model has to learn which dataset the presented expression is from to decide what latex normalization should be

followed. It is therefore likely that correctly normalizing all expression labels is required to allow the model to generalize better over the three datasets.

6.3 Custom tokenizer experiments

To evaluate which tokenizer has the best speed and accuracy combination we train models with the three previously described tokenizers MT1, MT4 and MT5 and compare them. The MT1 tokenizer is the tokenizer that simply uses the tokens that were specified in the MathWriting dataset, the MT4 tokenizer has some added tokens for frequently used functions and finally the MT5 tokenizer additionally has some tokens for frequently used combinations of syntactic characters. The results are shown in Table 3.

6.3.1 Results and discussion

Table 3: Tokenizer Comparison Results

Method Name	MW Validation Accuracy (↑)	MW Validation CER (↓)	GFLOPs (↓)
Previous model (original tokenizer)	0.87	2.50	58.56
SMALL + MT1	0.88	2.31	57.08
SMALL + MT4	0.88	2.33	57.08
SMALL + MT5	0.88	2.36	52.50

As shown in Table 3 the CER and especially the accuracy metric are only slightly affected by the tokenizer choice. The differences between the GFLOP requirements are mostly as expected, with the exception that the MT1 and MT4 tokenizers resulted in the same amount of GFLOPs. The reason for this is that the chosen test-expressions for the GFLOPs calculation (listed in section 5.3) do not include any occurrences of the tokens that were added to the MT4 tokenizer. This shows that more test-expressions should have been used to make the metric more reliable.

6.4 Tree-based inference approach

In the next experiment, a model is trained for the tree-based inference method and compared to the previous model. The GFLOPs are not measured in this experiment, as it has only been implemented for the normal sequential decoding inference.

6.4.1 Results and discussion

Table 4: Tree-based Inference Results: This table shows the accuracy metrics of the tree based inference experiment. The previous SMALL + MT1 run is also shown in this table to allow for easier comparison.

Method Name	MW Validation Accuracy (↑)	MW Validation CER (↓)
SMALL + MT5 (previous best)	0.88	2.36
SMALL + MT1	0.88	2.31
SMALL + MT1 + tree-based inference	0.84	3.20

The results in Table 4 show that the tree-based inference approach achieved 4 percent points lower accuracy and 9 percent points more CER than its non-tree based counterpart. This might be due to the training method used in the tree based approach. Unlike the training method that is used for the non-tree based models, it puts much more weight on earlier tokens in the expression. This might mean that the model looses the ability to correctly predict long expressions, which would explain the decreased accuracy. A related issue might be that expressions with many branches occur more often than expressions with few branches, which complicates the analysis of these results even further. In conclusion, the tree-based approach might have potential, however its training process requires more fine-tuning before it can meet or exceed the accuracy of the non-tree based models.

6.5 Decoder Distillation experiments

In order to get good distillation results, an appropriate student model size has to be found. There are more distillation hyperparameter that can be tuned, such as learning rate and the ratio of soft and hard loss, however these are kept constant due to time constraints. A student model will be trained with [5, 4, 3, 2] decoder transformer layers and compared according to their MathWriting CER and GFLOP requirements. The results will be reported in the Table 5.

6.5.1 Results and Discussion

Table 5: Distillation Results on MW Dataset

Method Name	MW Validation Accuracy (↑)	MW Validation CER (↓)	GFLOPs (↓)
SMALL + MT5 (previous best, teacher)	0.88	2.36	52.50
5 layers	0.82	3.33	47.90
4 layers	0.81	3.36	43.31
3 layers	0.80	3.48	38.71
2 layers	0.79	3.93	34.11

In the results shown in Table 5 it can be seen that for each decoder-layer that is removed, the model uses 4.6 GFLOPs less. For the distilled models it is also evident that each layer that is removed decreases the accuracy almost exactly linearly as well. Interestingly, the CER metric does not follow this trend: the CER gap between 5 and 4 layers is much smaller than the CER gap between 3 and 2 layers. This might show that having more decoder layers allow the model to recover from a wrongly predicted token better than with fewer decoder layers. The most important observation that can be made from these results is that the accuracy difference between the largest distilled model and the teacher model seems to be notably high. Its accuracy scores are closer to the much smaller 2 decoder layer model than to the teacher model, which is an indication that the distillation process could be improved. Apart from this, these results are quite expected: less decoder layers means that the model has less capacity overall and can therefore not make as precise predictions. It is not obvious which of these models can be considered to be the best, since even though the smallest model has a much higher CER, it also requires much fewer GFLOPs.

6.6 Convergence behavior plot

Figure 10: The validation accuracy, CER and loss history of the SMALL MT4 model training

Figure 10 shows the typical accuracy and loss curve of the training runs: They very quickly an acceptable accuracy and then continue slowly increase. It seems like the models might be able to still improve by a few percentage points with more training, since there is a slight upwards trend even towards the end of the training run. These plots also show that towards the end of the run the three metrics do not behave exactly the same: the loss slightly increases, the CER seems to converge to around 2.8 whereas the accuracy still seems to gradually improve.

6.7 Final compilation of methods

Table 6: This table shows how each added method makes the model more accurate or less computationally expensive

Method Name	MW Validation Accuracy (↑)	MW Validation CER (↓)	GFLOPs (↓)
BASE	0.91	2.13	553.34
SMALL	0.87	2.50	58.56
SMALL + MT5	0.88	2.36	52.50
DISTILLED + MT5	0.79	3.93	34.11

As seen in Table 6, the final distilled model makes around twice as many errors however it uses less than a tenth of GFLOPs. If GFLOP cost valued less, the undistilled version of the small model using the MT5 tokenizer is the best tradeoff between accuracy and computational cost.

6.8 Failure case analysis

In order to demonstrate the limitations of this model, some examples of expressions that are not recognized correctly are listed below.

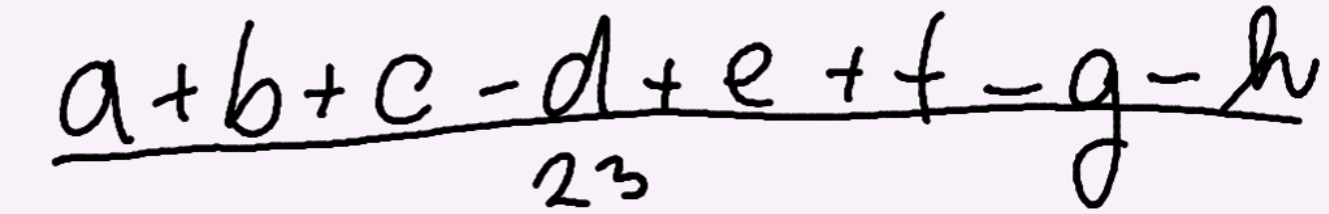


Figure 11: This expression is recognized as: $\frac{a+b+c-d+e+t-h}{2^3}$

As seen in Figure 11 if there is one error the rest of the expression usually also has a higher error rate. This could be due to the decoder being trained in a way that "assumes" that the part of the expression that was decoded so far was correct.

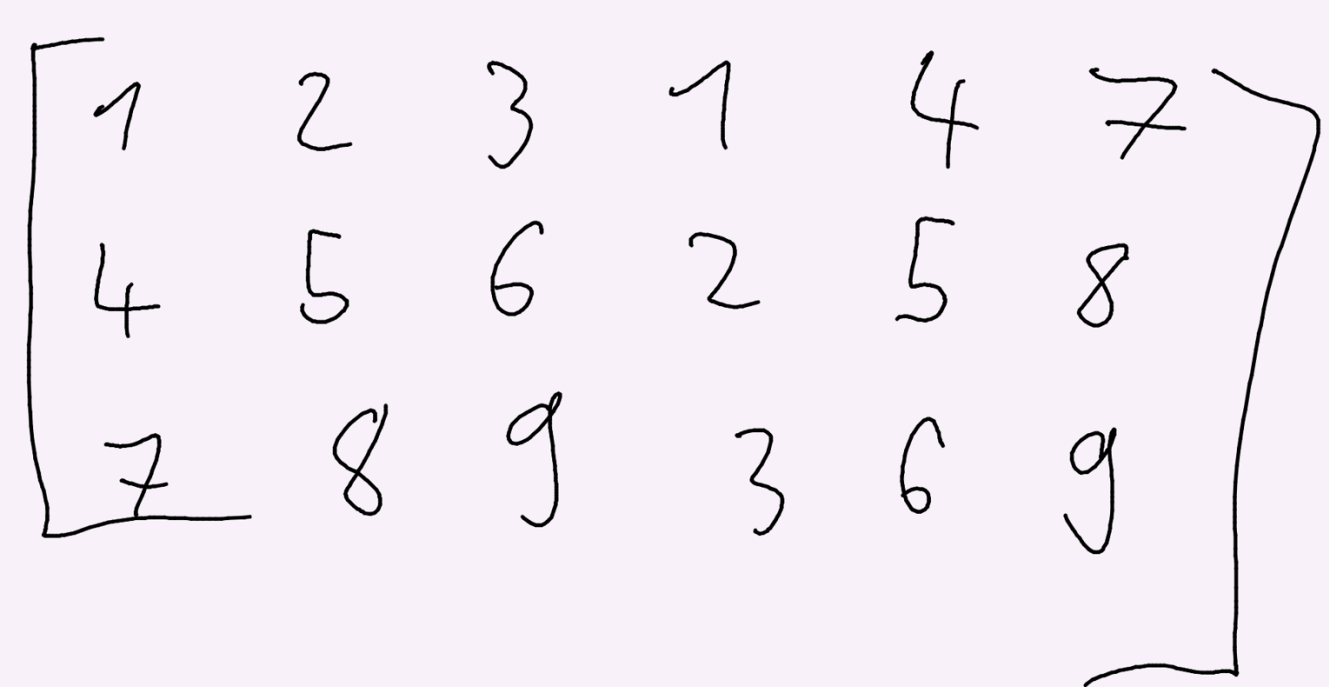


Figure 12: This expression is recognized as: $\begin{bmatrix} 1 & 2 & 2 & 1 & 4 & 7 \\ 4 & 6 & 2 & 2 & 58 & \\ 7 & 8 & 3 & 36 & & \end{bmatrix}$

Figure 12 shows that the model struggles with large matrices. It is interesting that the model is able to recognize such large matrices at all, since the dataset likely hardly contains matrices of this size.

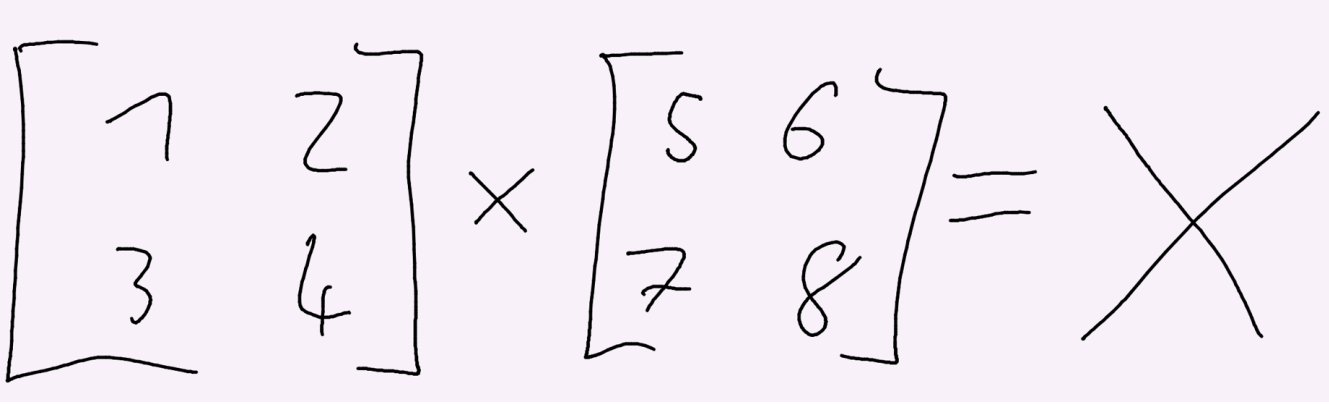


Figure 13: This expression is recognized as: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 56 & \\ 7 & 8 \end{bmatrix} = X$

Figure 13 demonstrates that the model often misrecognizes expressions that contain several matrices.

Figure 14: This expression is recognized as: $f(x) = 12x + 1$

As seen in Figure 14 the model does not have a concept of what it means when a part of the expression is crossed out. In this case it recognized the crossed out part of the expression, however more often the expression is recognized completely wrong if some part is crossed out. Making a model that can handle these cases as a human could is challenging, since no datasets has many examples of such cases.

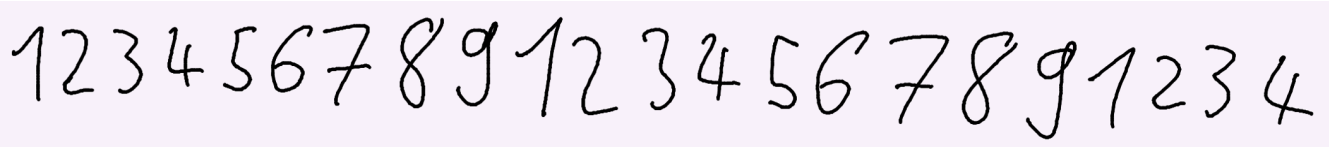


Figure 15: This expression is recognized as: 12467891234567891234

Long expressions, even if they appear to be simple, can cause the model to make errors even at the beginning of the expression.

7 Further Research

There are many ways how the speed and accuracy of this model could be improved further. Since this thesis was mostly focused on the decoder, the encoder likely still has much potential for improvements. Since the decoder distillation has been shown to be very effective, distilling the encoder model will most likely also work well. Another interesting extension would be to use a variable aspect ratio vision encoder model, as the current approach of stretching the input image to be square might not be ideal.

Furthermore, interesting extensions to the TrOCR model such as the Decoder-only DTrOCR [Fuj23] can also be used as base models for the HMER task, which might bring an accuracy improvement.

One improvement that is relatively easy to implement is early stopping. At the moment much training time is spent on runs that almost fully converged long before that maximum iteration count was reached. This training time could be spent better; for example for more variety in the experiments. Some models also exhibit clear signs of overfitting, which could also be addressed by early stopping. However, it is not trivial to decide when to stop: in many runs, the validation-loss

metric starts to increase after some time, even though the accuracy and CER validation metrics continue to improve slightly.

Finally the model could be quantized to a lower numerical precision to reduce real-world cpu time without changing the number of GFLOPs.

8 Conclusion

The original goal of this thesis was to decrease the computational cost of an HMER model without compromising on prediction accuracy. In this thesis I have implemented several methods to achieve this, including distillation, trying a tree-based inference approach and using a tokenizer that allows the model to be much smaller. In conclusion, the original goal of decreasing the computational cost of the model has been achieved with an acceptable decrease in prediction quality. Several answers for the research questions can be given: The computational efficiency of HMER models can be improved by using appropriately sized models, adjusting the tokenizers to better fit the task and by distilling the final model.

References

[And67] Robert H. Anderson. Syntax-directed recognition of hand-printed two-dimensional mathematics. In *Interactive Systems for Experimental Applied Mathematics - Proceedings of the Association for Computing Machinery Inc. Symposium, ACM 1967*, 1967.

[Fuj23] Masato Fujitake. DTrOCR: Decoder-only Transformer for Optical Character Recognition. 8 2023.

[GFM24] Philippe Gervais, Anastasiia Fadeeva, and Andrii Maksai. MathWriting: A Dataset For Handwritten Mathematical Expression Recognition. 4 2024.

[GLSY24] Tongkun Guan, Chengyu Lin, Wei Shen, and Xiaokang Yang. PosFormer: Recognizing Complex Handwritten Mathematical Expression with Position Forest Transformer. 7 2024.

[hug] Transformers — hugging face. `https://huggingface.co/transformers`. Accessed: 2025-06-30.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. 3 2015.

[KJGK24] Thennal D K, Jesin James, Deepa P Gopinath, and Muhammed Ashraf K. Advocating Character Error Rate for Multilingual ASR Evaluation. 10 2024.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[LLC+23] Minghao Li, Tengchao Lv, Jingye Chen, Lei Cui, Yijuan Lu, Dinei Florencio, Cha Zhang, Zhoujun Li, and Furu Wei. TrOCR: Transformer-Based Optical Character Recognition with Pre-trained Models. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence, AAAI 2023*, volume 37, 2023.

[LLG+22] Chenxia Li, Weiwei Liu, Ruoyu Guo, Xiaoting Yin, Kaitao Jiang, Yongkun Du, Yuning Du, Lingfeng Zhu, Baohua Lai, Xiaoguang Hu, Dianhai Yu, and Yanjun Ma. PP-OCRv3: More Attempts for the Improvement of Ultra Lightweight OCR System. 6 2022.

[LN16] Anh Duc Le and Masaki Nakagawa. A system for recognizing online handwritten mathematical expressions by using improved structural analysis. *International Journal on Document Analysis and Recognition*, 19(4), 2016.

[Mica] Microsoft. microsoft/trocr-base-handwritten. `https://huggingface.co/microsoft/trocr-base-handwritten`. Accessed: 2025-06-30.

[Micb] Microsoft. microsoft/trocr-small-handwritten. `https://huggingface.co/microsoft/trocr-small-handwritten`. Accessed: 2025-06-30.

[MPA+21] Rachit S. Munjal, Arun D. Prabhu, Nikhil Arora, Sukumar Moharana, and Gopi Ramena. STRIDE: Scene Text Recognition In-Device. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2021-July, 2021.

[Pea] Pearson. Aida calculus math handwriting recognition dataset. `https://www.kaggle.com/datasets/aidapearson/ocr-data/data`. Accessed: 2025-06-30.

[Pho] Photomath LLC. Photomath. `https://photomath.com/`. Accessed: 2025-06-30.

[PyT] PyTorch Development Team. Pytorch: An imperative style, high-performance deep learning library. `https://pytorch.org`. Accessed: 2025-06-30.

[Rom] Romeo Sommerfeld. Trocr training implementation. https://github.com/rsommerfeld/trocr.

[XMSL+23] Yejing Xie, Harold Mouchère, Foteini Simistira Liwicki, Sumit Rakesh, Rajkumar Saini, Masaki Nakagawa, Cuong Tuan Nguyen, and Thanh-Nghia Truong. ICDAR 2023 CROHME: Competition on˜Recognition of˜Handwritten Mathematical Expressions. In *Lecture Notes in Computer Science*, volume 14188 of *Lecture Notes in Computer Science*, pages 553–565, San josé, CA,, United States, 8 2023. Springer Nature Switzerland.

[YTD+21] Ryota Yoshihashi, Tomohiro Tanaka, Kenji Doi, Takumi Fujino, and Naoaki Yamashita. Context-Free TextSpotter for Real-Time and Mobile End-to-End Text Detection and Recognition. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12822 LNCS, 2021.

[ZGY+21] Wenqi Zhao, Liangcai Gao, Zuoyu Yan, Shuai Peng, Lin Du, and Ziyin Zhang. Handwritten Mathematical Expression Recognition with Bidirectionally Trained Transformer. 5 2021.

MT1 Vocabulary

Below is the complete vocabulary of the MT1 tokenizer sorted alphabetically. Note that the first element is a space. This vocabulary is based on the MathWriting tokens as described in Appendix I of [GFM24].

	!	&	(	)
*	+	,	-	.
/	0	1	2	3
4	5	6	7	8
9	:	;	<	</s>
<mask>	<pad>	<s>	<unk>	=
>	?	A	B	C
D	E	F	G	H
I	J	K	L	M

```
N                   O                   P                   Q                   R
S                   T                   U                   V                   W
X                   Y                   Z                   [                   \#
\%                  \&                  \Delta              \Gamma              \Lambda
\Leftrightarrow     \Omega              \Phi                \Pi                 \Psi
\Rightarrow         \Sigma              \Theta              \Upsilon            \Vdash
\Xi                 \\                  \_                  \aleph              \alpha
\angle              \approx             \backslash          \begin{matrix}      \beta
\bigcap             \bigcirc            \bigcup             \bigoplus           \bigvee
\bigwedge           \bullet             \cap                \cdot               \chi
\circ               \cong               \cup                \dagger             \delta
\div                \dot                \emptyset           \end{matrix}        \epsilon
\equiv              \eta                \exists             \forall             \frac
\gamma              \ge                 \gg                 \hat                \hbar
\hookrightarrow     \iff                \iint               \in                 \infty
\int                \iota               \kappa              \lambda             \langle
\lceil              \le                 \leftarrow          \leftrightarrow     \lfloor
\ll                 \longrightarrow     \mapsto             \mathbb             \mathbb{A}
\mathbb{B}          \mathbb{C}          \mathbb{D}          \mathbb{E}          \mathbb{F}
\mathbb{G}          \mathbb{H}          \mathbb{I}          \mathbb{J}          \mathbb{K}
\mathbb{L}          \mathbb{M}          \mathbb{N}          \mathbb{O}          \mathbb{P}
\mathbb{Q}          \mathbb{R}          \mathbb{S}          \mathbb{T}          \mathbb{U}
\mathbb{V}          \mathbb{W}          \mathbb{X}          \mathbb{Y}          \mathbb{Z}
\models             \mp                 \mu                 \nabla              \ne
\neg                \ni                 \not                \notin              \nu
\odot               \oint               \omega              \ominus             \oplus
\otimes             \overline           \partial            \perp               \phi
\pi                 \pm                 \prime              \prod               \propto
\psi                \rangle             \rceil              \rfloor             \rho
\rightarrow         \rightleftharpoons  \sigma              \sim                \simeq
\sqrt               \sqsubseteq         \subset             \subseteq           \subsetneq
\sum                \supset             \supseteq           \tau                \theta
\tilde              \times              \top                \triangle           \triangleleft
\trianglelefteq     \underline          \upsilon            \varphi             \varpi
\varsigma           \vartheta           \vdash              \vdots              \vec
\vee                \wedge              \xi                 \zeta               \{
\|                  \}                  ]                   ^                   _
a                   b                   c                   d                   e
f                   g                   h                   i                   j
k                   l                   m                   n                   o
p                   q                   r                   s                   t
u                   v                   w                   x                   y
z                   {                   |                   }
```

Training Hyperparameters

Number of iterations: 1_000_000
Learning Rate: 2.5e-5