TRANSCRIPTION OF MATHEMATICAL EXPRESSIONS INTO TYPST SYNTAX USING A VISION-ENCODER-DECODER ARCHITECTURE

Jeppe Klitgaard (S250250)

ABSTRACT

Deep learning has revolutionized Optical Character Recognition, yet transcribing mathematical expressions remains a significant challenge due to complex spatial layouts and rigid syntax requirements. This paper presents *Typstscribe*, a vision-encoder-decoder model adapted from the TrOCR framework to transcribe mathematical images into the modern Typst markup language. To address the scarcity of domain-specific data, we introduce a synthetic dataset of 3.3 million image-label pairs generated via LaTeX-to-Typst conversion. A central contribution of this work is the development of a tailored tokenizer that treats Typst keywords as atomic units, thereby enhancing the model's inductive bias and semantic coherence compared to standard Byte-Pair Encoding. Although computational constraints limited final convergence, our experiments demonstrate the advantages of specialized tokenisation for syntax-aware transcription tasks.

1. INTRODUCTION

Deep learning models have made significant advances in traditional Optical Character Recognition (OCR) challenges in recent years [1], [2], and are thus good candidates to tackle the difficult problem of machine transcription of mathematical expressions, which often have concise, spatially complex layouts and place significant semantic importance on small visual features. Unlike standard language text, mathematical notation lacks a universal Unicode representation, necessitating transcription into rigid syntaxes such as those offered by LaTeX, Typst, or MathML[3].

This report describes the motivation and implementation of the *Typstscribe* deep learning model for transcription of mathematical expressions from images into the Typst markup language. We place particular emphasis on the development of a tailored tokenizer which enhances the inductive bias of the model by alignment with the structure of the Typst syntax.

2. THE TYPST LANGUAGE

The Typst [4] markup language is a modern alternative to established typesetting systems such as TeX/LaTeX as well as document processors such as Microsoft Word and Google Docs. It has a strong focus on usability and performance while maintaining a high typographic quality. Additionally, Typst offers modern features such as online collaboration, accessibility support, native scripting, and extensibility through packages.

As an example of a mathematical expression typeset in Typst, consider Listing 1:

```
$                                                    Typst
underbrace(f(theta), "obj") = cases(
  a b [theta/2] & "if" norm(bold(v)) >= 1,
  integral_(cal(M)) r dif theta & "else"
)
$
```

Listing 1. Mathematical expression typeset in Typst.

With the LaTeX equivalent seen in Listing 2 and their renders shown in Table 1:

```
\[                                                   LaTeX
\underbrace{f(\theta)}_{\text{obj}} = \begin{cases}
  ab \left[ \frac{\theta}{2} \right] & \|
  \boldsymbol{v}\| \ge 1 \\
  \int_\mathcal{M} r \, d\theta & \text{else}
\end{cases}
\]
```

Listing 2. Mathematical expression typeset in LaTeX.

Typst	LaTeX
$\underbrace{f(\theta)}_{\text{obj}} = \begin{cases} ab\left[\frac{\theta}{2}\right] & \text{if } \|\boldsymbol{v}\| \geq 1 \\ \int_{\mathcal{M}} r \, \mathrm{d}\theta & \text{else} \end{cases}$	$\underbrace{f(\theta)}_{\text{obj}} = \begin{cases} ab\left\lfloor\frac{\theta}{2}\right\rfloor & \|\boldsymbol{v}\| \geq 1 \\ \int_{\mathcal{M}} r \, d\theta & \text{else} \end{cases}$

Table 1. Side-by-side comparison of the Typst and LaTeX renderings of Listing 1 and Listing 2 respectively.

Notable differences between the two systems include the improved ergonomics of the Typst syntax as well as features such as automatic fractions, delimiter sizing, and ligatures. Furthermore, the Typst syntax in Listing 1 compiles as-is, whereas the LaTeX equivalent requires use of the `amsmath` package as well as additional boilerplate code as seen in Appendix A.1.

3. DATA

Due to the relative recency of the Typst language, there exist only very limited publicly available datasets, which are unlikely to be suitable or sufficient for training a deep learning transcription model. To address this scarcity, we spent substantial effort generating a dataset by leveraging the `tex2typ` program developed by Typst community member `ParaN3xus` [5], which is capable of generating Typst expression from LaTeX input. With this, we are able to construct a dataset of Typst expressions and associated images by conversion of the data found in the `fusion-image-to-latex-datasets` dataset [6] to produce a dataset of 3.3 million image-label pairs which is publicly available at [7]. In order to facilitate efficient random-access loading during training, the dataset is made available in the `WebDataset` format as opposed to the original RAR archival format.

4. VISION-ENCODER-DECODER MODEL

The model architecture used in this project is based on the TrOCR model developed by Microsoft [1], which has been shown to work for both classic text recognition as well as transcription into markup [8], [9], [10] with varying degrees of success.

We leverage a pre-trained model, `microsoft/trocr-small-stage1` [1], whose architecture is depicted in Fig. 1 and comprises a DeIT Vision Transformer [11]. The model first resizes images to $H \times W = 384 \times 384$ pixels and subsequently splits them into $P \times P = 16 \times 16$ patches. These are then flattened and linearly projected onto a $D = 384$-dimensional embedding space alongside positional embeddings which describe the patch positions in the full, resized image.

This sequence of embeddings, augmented with a classification token `[CLS]` and a distillation token `[DIST]`, is passed through 12 layers of hidden size D, each consisting of a 6-head self-attention block and a fully-connected feed-forward network. The embeddings are expanded onto a $4D = 1536$-dimensional space prior to activation using a GELU activation function before being projected back to the original D-dimensional space. Finally, the model features residual connections and layer normalisations as described in [11].

Rather than the classification heads used in the original DeIT model, we pass the embeddings from the vision encoder into a pretrained MiniLM Transformer described in [12], to which an encoder-decoder cross-attention mechanism enabling the retrieval of visual information from queries in the decoder is added. This cross-attention mechanism is placed between the standard 8-head self-attention and the feed-forward network blocks that make up the 6 layers of the decoder. The hidden states from the decoder are projected onto a vocabulary of size $V = 4096$ over which probabilities are computed using a softmax function, bringing the total parameter count to approximately 62 million. Finally, beam search is used to generate the output sequences from the obtained probabilities. Generation of the vocabulary and tokenisation is described further in Section 5.

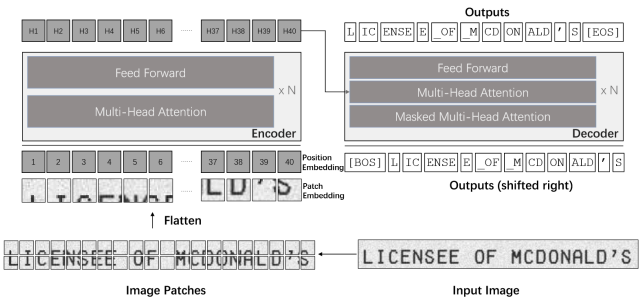

Fig. 1. The TrOCR model architecture showing the Vision Transformer encoder and Transformer decoder components. Figure adapted from [1] under CC BY-NC-SA 4.0.

4.1. Comparison with other architectures

Alternative model architectures include a combination of Convolutional and Recurrent Neural Networks (CNN, RNN), which may be used in a similar encoder-decoder configuration to the TrOCR model described above, as proposed in [2]. We theorise that such architectures are ill-suited for transcription into rigid markup languages such as Typst due to their limited ability to capture long-range dependencies required by the syntax structure of such languages. As an example, the opening and closing parentheses of the `cases` element in Listing 1 are not reflected readily in the visual representation of the expression. This may in part be mitigated through the addition of attention mechanisms or LSTM networks, the discussion of which is beyond the scope of this report.

5. TOKENISATION

The default vocabulary and associated tokenisation scheme used in the TrOCR architecture is based on the Byte-Pair Encoding (BPE) [13] and SentencePiece [14] algorithms with a vocabulary size of $V = 64044$. We propose that a significantly smaller vocabulary specifically tailored to the Typst language may yield efficiency and performance benefits due to the constrained syntax of the language and the intended application of the model. To this end, we construct a custom BPE tokenizer with a vocabulary size of $V = 4096$ using the `tokenizers` library by HuggingFace. Given the limited number of symbols in the Typst Language, the tokenizer is initialised with a base vocabulary consisting of the functions exposed in its math-mode syntax as documented on the website [15] as well as the `sym.txt` table of the Codex project [16], which contains ASCII mappings to a vast number of unicode symbols.

Furthermore, we employ a pre-tokenisation step in which the individual digits of numbers are split into separate tokens in order to encourage the model to learn the appropriate structure rather than the memorisation of specific numbers.

Lastly, the custom tokenizer is trained on the corpus of 3.3 million Typst expressions found in the dataset described in Section 3 using a BPE training procedure with a target vocabulary size of 4096. To visualise the difference in tokenisation between our custom Typst tokenizer and the default TrOCR tokenizer, we tokenize the expression `underbrace(f(theta), "obj")` using both to produce:

Ours: `underbrace` `(` `f` `(` `theta` `)` `,` `"` `o` `b` `j` `"` `)`

Default: `under` `b` `race` `(` `f` `(` `the` `ta` `),` `"` `ob` `j` `")`

Coincidentally, both methods produce sequences of length 13. However, our tokenizer offers higher semantic coherence despite a smaller vocabulary. By encoding Typst keywords as atomic tokens, we imbue the model with a stronger inductive bias and disambiguate the output. As an example of this, we note that the default tokenizer cleaves the `underbrace` function into 3 tokens: `under`, `b`, `race`, which may lead the model to generate invalid syntax in situations of uncertainty. Similarly, the smaller vocabulary size reflects the limited active corpus of the Typst language compared to general natural language text, introducing further structural consistency and lessening the training required.

Our tokenizer is made publicly available on HuggingFace [17].

6. MODEL TRAINING

While the use of pre-trained weights from the `microsoft/trocr-small-stage1` model offers a strong starting point and significantly reduces the training burden compared to initialisation from random weights, the invasive surgery of replacing the original tokenizer complicates the fine-tuning process significantly. Not only have the tokens changed their meaning entirely, the change of vocabulary size V requires that the hidden states within the decoder must be projected onto a different space than that seen during pre-training and the weights of the decoder must thus be partially reinitialised. Doing so naïvely may lead to instability during training and could end up damaging the network of the image encoder. To circumvent this, we *freeze* the weights of the encoder during the initial training phase — here referred to as *Stage 2* — allowing the decoder to adapt to the new vocabulary without perturbing the learned image representations. In the final stage, *Stage 3*, the encoder is unfrozen and trained jointly with the decoder in order to produce the final model, which is evaluated in Section 8. A relative warm-up period of 10% was used alongside a standard cross-entropy loss for all training.

Another notable model from our training experiments is the default pre-trained model without modification of the tokenizer. Such a model was fine-tuned on the dataset according to the parameters in Table 2 in order to evaluate the performance of the change of tokenizer.

We note that both models were trained on a subset of the full dataset consisting of approximately 900,000 samples due to storage constraints on the hardware used for training.

Parameter	Custom, Stage 2	Custom, Stage 3	Default
Epochs	2	4	6
Learning Rate	E: N/A D: 5×10^{-5}	E: 3×10^{-5} D: 5×10^{-5}	E: 3×10^{-5} D: 5×10^{-5}

Table 2. Training parameters of the different models and stages. All learning rates used a cosine annealing schedule with 10% warm-up.

Training was instrumented using the `Seq2SeqTrainer` from the `transformers` library and undertaken on a workstation equipped with an NVIDIA RTX 4090 GPU with 24 GB of VRAM provided by NanoNord A/S.

7. EVALUATION METRICS

In order to evaluate the performance of the model, we employ the familiar Character Error Rate (CER) in addition to two custom metrics, the first of which is the *compilability rate* defined as the fraction of evaluated predictions that successfully compile using the Typst compiler. While insufficient on its own, this metric offers valuable observability into the training process.

Since both the label and prediction are sequences of Typst code, we may render both expressions and compare them using either a Structural Similarity Index Measure (SSIM) or Intersection-over-Union (IoU) metric. While the latter may generally be considered a difficult metric, it is well-suited here given its ability to express the *semantic* similarity between Typst expressions, which will produce a score of unity for equivalent expressions. As such, it pairs well with CER, which captures the *syntactic* similarity between the sequences.

8. MODEL PERFORMANCE

Due to time and computational resource limitations, longer training runs were not possible. This is expected to have severely affected the achieved performance of both the model using the default tokenizer and, in particular, the *Typstscribe* model with custom tokenisation. This is reflected in Table 3, where our models are compared against others from the literature. Notably, an earlier model with a development version of our tokenizer and a larger TrOCR model was afforded significant training time and achieved an improved CER score of 0.5514, but was ultimately abandoned in favour of the architecture described above due to the infeasibly long training times required given the hardware available. As such the presented model performances do not necessarily reflect

limitations of the architecture or dataset, but rather training times that were not sufficient to achieve adequate learning.

This is supported by the fact that the Pix2Text model also leverages the TrOCR architecture and achieves a best-in-class CER score of 0.1674, with proprietary versions of the model reportedly being even lower. These models are reported to use training sets with millions of samples to train a similar parameter count as the one present in our models. Similarly, [8] report requiring several days of training time on a T4 GPU with a single epoch requiring upwards of 10 hours of computation.

Model	Output	CER	IoU	Compilable [%]
Typstscribe	Typst	0.7135	0.02311	35.94
TrOCR Fine Tune	Typst	0.7466	0.01812	24.22
Texify	LaTeX	0.7915	—	—
Latex-OCR	LaTeX	0.4479	—	—
Chhadia, Gupta [8]	LaTeX	0.515	—	—
Pix2Text V1.0 MFR [10]	LaTeX	0.1674	—	—

Table 3. Achieved model performances compared against models from the literature. CER statistics from our models were obtained from [8], [10].

As an example of the evaluation strategy employed, we consider the first sample from the evaluation data set shown in Table 4, in which the model demonstrates its ability to discern bold and upright characters, but struggles to correctly match delimiters, which explains the relatively low fraction of compilability presented in Table 3. Balancing delimiters is a particularly difficult task, which requires an attentive model with considerable understanding of the target syntax.

Additionally, we note the superfluous use of spaces and parenthesis in the ground-truth label, which arises from the generation of Typst code from LaTeX whereby the `tex2typ` library conservatively produces these to ensure that the Typst output is semantically equivalent to the parsed LaTeX syntax. Given the difficulty with which the model handles parenthesis, we suggest that the output may be improved by either mending the dataset to reduce the occurence of unnecessary parentheses and spaces, or alternatively performing a fine-tuning step after the initial training on a smaller dataset consisting of idiomatic Typst code. As an example, the idiomatic label associated with the sample in Table 4 might be `abs(bold(upright(u)))_1 < eta_4`.

Image	$\lvert\mathbf{u}\rvert_1 < \eta_4$
Truth	`| bold( upright( u ) ) | _ ( 1 ) < eta _ ( 4 )`
Pred.	`|bold (upright u ) _ (1 )<eta (4`

Table 4. Prediction and ground truth label of the first sample in the evaluation dataset.

9. CONCLUSION AND FUTURE WORK

We find that the obtained models following the TrOCR architecture do not perform adequately to be useful for the purposes of transcription. While the models appear able to learn the mapping between images onto different characters and operators in the Typst language, they are severely limited by their ability to correctly identify and locate delimiters required to construct valid outputs.

There is not sufficient evidence to suggest that this limitation arises from the architecture itself and we propose that vastly improved results may be attainable given the use of the full dataset and longer training times. While specialisation onto a limited domain of natural language text or depiction may be achieved with a relatively small amount of fine-tuning, we theorise that the training burden required to specialise the visual encoder to mathematical notation is larger than first anticipated in part due to the importance of minute features in the images and the lack of relevant examples in the dataset of the pre-trained image model.

The proposed tokenizer significantly improved the representation of Typst code by using a succinct and specialised vocabulary, which was used effectively by the model. It may be of interest to further specialise the tokenizer to yield an Abstract Syntax Tree, which could further improve the inductive bias of the model and help mitigate against improperly placed delimiters and arguments to the function calls in the produced code.

We propose that future work on a TrOCR model for mathematical transcription investigate whether larger models, such as those based on the `trocr-base-stage1` variant, are able to achieve improved performance. Additionally, we encourage such work to also include an investigation into the performance of the model proposed in this document given longer training times.

The code used to generate the data and train the models described in this document is made available on GitHub at `https://github.com/JeppeKlitgaard/DTU-02456-Deep-Learning-Project`.

10. REFERENCES

[1] M. Li *et al.*, "TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models." Accessed: Nov. 08, 2025. [Online]. Available: http://arxiv.org/abs/2109.10282

[2] B. Shi, X. Bai, and C. Yao, "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition." Accessed: Dec. 04, 2025. [Online]. Available: http://arxiv.org/abs/1507.05717

[3] World Wide Web Consortium, "Mathematical Markup Language (MathML) Version 3.0 2nd Edition." Accessed: Dec. 05, 2025. [Online]. Available: https://www.w3.org/TR/MathML/

[4] "Typst: The New Foundation for Documents." Accessed: Nov. 08, 2025. [Online]. Available: https://typst.app/

[5] "ParaN3xus/Tex2typ: LaTeX Math Equations to Typst Equations Conversion.." Accessed: Nov. 08, 2025. [Online]. Available: https://github.com/ParaN3xus/tex2typ

[6] "Hoang-Quoc-Trung/Fusion-Image-to-Latex-Datasets · Datasets at Hugging Face." Accessed: Nov. 08, 2025. [Online]. Available: https://huggingface.co/datasets/hoang-quoc-trung/fusion-image-to-latex-datasets

[7] Jeppe Klitgaard, "JeppeKlitgaard/Typst-Image-Dataset · Datasets at Hugging Face." Accessed: Dec. 04, 2025. [Online]. Available: https://huggingface.co/datasets/JeppeKlitgaard/typst-image-dataset

[8] N. Chhadia and S. Gupta, "Adaptation of OCR Models for LATEX Vision."

[9] L. Blecher, "Lukas-Blecher/LaTeX-OCR." Accessed: Nov. 08, 2025. [Online]. Available: https://github.com/lukas-blecher/LaTeX-OCR

[10] Breezedeus, "Pix2text-Mfr." Accessed: Dec. 01, 2025. [Online]. Available: https://huggingface.co/breezedeus/pix2text-mfr

[11] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training Data-Efficient Image Transformers & Distillation through Attention." Accessed: Dec. 03, 2025. [Online]. Available: http://arxiv.org/abs/2012.12877

[12] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers." Accessed: Dec. 03, 2025. [Online]. Available: http://arxiv.org/abs/2002.10957

[13] R. Sennrich, B. Haddow, and A. Birch, "Neural Machine Translation of Rare Words with Subword Units." Accessed: Dec. 04, 2025. [Online]. Available: http://arxiv.org/abs/1508.07909

[14] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, E. Blanco and W. Lu, Eds., Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. doi: 10.18653/v1/D18-2012.

[15] Typst Community, "Math – Typst Documentation." Accessed: Dec. 04, 2025. [Online]. Available: https://typst.app/docs/reference/math/

[16] Typst Community, "Codex Project." Accessed: Dec. 04, 2025. [Online]. Available: https://github.com/typst/codex

[17] Jeppe Klitgaard, "JeppeKlitgaard/Typst-Tokenizer · Hugging Face." Accessed: Dec. 05, 2025. [Online]. Available: https://huggingface.co/JeppeKlitgaard/typst-tokenizer

A. APPENDIX

A.1. Full LaTeX Sample Code

```LaTeX
\documentclass[8pt]{extarticle}
\usepackage[
   paperheight=36pt,
   paperwidth=108pt,
   top=2pt,
   bottom=2pt,
   left=2pt,
   right=2pt]{geometry}
\usepackage{amsmath}
\begin{document}
\pagestyle{empty}
\noindent
$\displaystyle
\underbrace{f(\theta)}_{\text{obj}} = \begin{cases}
  ab \left[ \frac{\theta}{2} \right] & \|\boldsymbol{v}\| \ge 1 \\
  \int_\mathcal{M} r \, d\theta & \text{else}
\end{cases}
$
\end{document}
```

DECLARATION OF USE OF GENERATIVE AI

This declaration **must** be filled out and included as the **final page** of the document. The questions apply to all parts of the work, including research, project writing, and coding.

- I/we have used generative AI tools: [yes]

If you answered *yes*, please complete the following sections. List the generative AI tools you have used:

- Google Gemini 2.5/3.0 Pro

Describe how the tools were used:

- The final draft of the report was passed in with instructions to provide feedback on the coherence and grammatical correctness of the report. The model identified a number of small grammatical errors and suggested alternatives for often-used words such as “additionally” and “lastly”. Some of these suggestions were incorporated into the final report, while others were disregarded. This was done a couple of times as the final draft was adjusted.
- The model was asked to turn a metric computation notebook cell into a function that worked with the `batched` keyword. The model produced a working function, which was subsequently modified to include the IoU and `ratio_compilable` metrics.
- The model was asked about how to turn the default tokenizer used in the pretrained TrOCR model into a `fast` tokenizer. Despite several attempts, a correct implementation was not produced and instead the tokenizer used in the “Default” model is left to use the non-fast tokenizer. The model output was not incorporated into the code.
- The model was asked to summarise how to log generated text predictions via Tensorboard. The model produced the `LogPredictionsCallback` class, an adapted version of which is used in `trainer.ipynb`.
- The model was asked to clarify the correct expression for the total number of steps of a training schedule given a number of epochs, a dataset length and the batch size as part of a transition from a random-access Dataset to the iterable IterableDataset class. The output was used to construct the step count passed into the trainer in `trainer.ipynb`:

```
EFFECTIVE_BATCH_SIZE = BATCH_SIZE * GRADIENT_ACCUMULATION_STEPS * 1 # 1 for number of GPUs
MAX_STEPS_STAGE2 = len(ds["train"]) * EPOCHS_STAGE2 // EFFECTIVE_BATCH_SIZE
```

- The model was asked to summarise the difference between DeIT and ViT models to aide in my understanding. This was not incorporated into the report.