Group 16

Zihan Wu (zihanw@seas.upenn.edu), Shukai Yin (sky1122@seas.upenn.edu)

Title: Texify

1. Summary

Texify is a web application designed to automatically convert handwritten mathematical content, such as equations, symbols, matrices, and tables, into clean LaTeX code. When users upload images or display formulas in front of their webcam, the system recognizes their layout and symbols and generates editable LaTeX code to reduce the time spent manually typing formulas.

The final system consists of three pipelines by first classifying whether the image is a formula, a matrix, or a table. Then the system will execute corresponding models to transform handwritten formulas, matrices, or tables into LaTex codes. The project bridges the gap between handwritten work and digital publishing by combining image processing, optical character recognition (OCR), and structured content parsing.

2. Problem Definition & Audience

Mathematical communication plays a central role in science, engineering, and quantitative disciplines. Although most final materials are prepared digitally using tools like LaTeX, the thinking and planning process often happens through handwritten notation - on paper, whiteboards, or tablets. Students sketch solutions in notebooks, researchers brainstorm equations on scratch paper, and instructors draft formulas on boards before converting them into lecture slides or formal documents.

Although tools like LaTeX offer precision and formatting flexibility, manually converting handwritten formulas into LaTeX code is tedious and error-prone. Even for experienced users, typing long equations, aligning matrices, or formatting complex expressions involves significant manual work. Moreover, existing OCR tools that perform handwritten math recognition are often expensive, closed-source, limited in scope, and fail to generalize to structured formats such as matrices or tables.

Our project, Texify, aims to develop a fast and accurate web app that converts handwritten mathematical content, including symbols, equations, matrices, and tables, into LaTeX codes. The intended users are students, researchers, instructors, and technical professionals. Users will upload an image of handwritten math or simply display it in front of their computer camera, and the application will automatically recognize both the symbols and their structural relationships, producing a syntactically correct LaTeX representation.

3. Solution and Pipeline

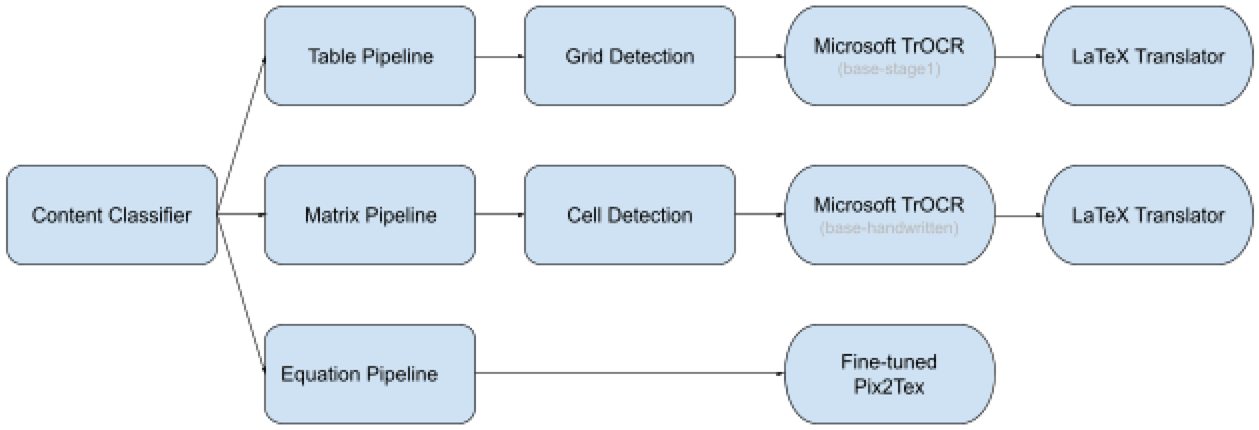


Figure 1: Texify main pipeline

Texify uses a modular pipeline to handle different types of handwritten mathematical content effectively. The process begins with a content classifier that identifies whether an uploaded image contains an equation, matrix, or table. This allows Texify to route the image into a specialized pipeline designed for that structure. For equations, the system relies on a fine-tuned Pix2Tex model, a transformer-based OCR that converts formula images directly into LaTeX sequences. Fine-tuning on CROHME (online handwritten math equations) datasets enables the model to handle diverse handwriting styles and layouts, while preprocessing steps like resizing and normalization ensure consistent inputs.

Tables and matrices require additional structural reasoning, so Texify combines traditional image processing with transformer OCR. The table pipeline detects grid lines through vertical and horizontal edge filtering, segments the cells, and uses Microsoft TrOCR to recognize content inside each cell. A custom LaTeX formatter then reconstructs the full tabular structure. The matrix pipeline follows a similar segmentation strategy but focuses on bracket shapes and uniform numeric layouts, producing LaTeX in matrix environments such as *bmatrix*. These specialized modules are integrated into a simple web interface where users can upload images and receive clean, editable LaTeX output. This design offers both flexibility and accuracy across different mathematical formats.

4. Baseline Equation Pipeline Results

Introduction

The baseline model was designed as an experimental prototype and proof of concept. Our goal was to implement and train a custom lightweight convolutional neural network (CNN) to recognize a small set of handwritten Greek letters and map them to LaTeX commands. This early-stage model allowed us to test whether a simple vision-only classifier could reliably identify individual symbols before dealing with full equations or more complex mathematical structures. The results from this baseline informed both our preprocessing choices and our decision to move toward more expressive sequence models later in the project.

Data

Selected greek symbols from hasyv2 dataset: {alpha, beta, chi, delta, epsilon, pi}.

Performance

Epoch 12 achieves best validation accuracy, which is 0.9449. Confusion appears mostly between visually similar shapes (e.g., *alpha* ↔ *delta*, *epsilon* ↔ *delta* tails). Precision/recall is greater than 0.95 for beta, chi, epsilon. delta shows lower precision (~0.79), while alpha and pi show lower recall (~0.81 and ~0.86 respectively).

Takeaway

The classifier, combined with robust preprocessing, already reaches strong performance on the held-out test set. Remaining errors are concentrated in symbol pairs that are shape-similar or stylistically ambiguous. A key limitation is the mismatch between training and inference data: the training samples are "clean" 224×224 patches (perfect black ink on white background), while real inputs are larger (e.g., 2556×1179) and contain noise, uneven lighting, and variable stroke thickness. To mitigate this domain shift, we require substantial preprocessing at inference time—center cropping, resizing to a square input, and contrast enhancement. These observations motivated us to (1) expand the symbol vocabulary and (2) eventually move beyond isolated classification toward a more flexible sequence model.

5. Results of Development Iterations

Modified Equation Pipeline Results:

Introduction

The modified pipeline builds directly on the baseline CNN classifier but extends its scope from a small set of Greek letters to a richer vocabulary of mathematical symbols. The goal of this stage was to test whether our lightweight convolutional architecture could still maintain high accuracy when tasked with distinguishing a larger, more heterogeneous symbol set that includes logical operators, set relations, and arithmetic symbols. This experiment helped us understand the scalability of the symbol-level approach and its practicality for supporting real equation vocabularies.

Data

Selected symbols from hasyv2 dataset:

- Greek symbols: {alpha, beta, chi, delta, epsilon, pi}.
- Math symbols: {not-equal, approximate, equivalent, plus-or-minus, and, therefore, because, Rightarrow, in, not-in, divide, exists, for-all, greater-than-or-equal-to, infinity, less-than-or-equal-to, real number, integers, sum}

Performance

Epoch 11 achieves the best validation accuracy at 0.9925. The expanded symbol classifier remains very strong and stable across categories: most symbols reach F1-scores above 0.98, with many near 1.00. The original Greek symbols (α, β, χ, δ, ε, π) maintain perfect precision and recall, indicating that scaling up did not harm earlier performance. Newly added symbols such as $\approx$, $\wedge$, $\therefore$, $\exists$, $\forall$, ∞, $\mathbb{R}$, $\mathbb{Z}$, and $\in$ also perform near-perfectly, while symbols like $\equiv$, $\pm$, $\Rightarrow$, $\div$, $\geq$, $\notin$, and $\sum$ are slightly weaker but still highly reliable. The main challenges are $\neq$ and $\leq$ (F1 around 0.92 and 0.96), where visual similarity to other horizontal-stroke symbols and handwriting variation cause occasional errors. Overall, the classifier meets or exceeds the 95% per-class accuracy target for almost all symbols, making it a solid basis for later structural and layout modeling.

Takeaway

Extending the vocabulary confirms that our CNN-based classifier can scale to a fairly large set of mathematical symbols while preserving high validation accuracy. This suggests that, for isolated symbol recognition, a lightweight supervised model plus careful preprocessing is sufficient and robust. However, this approach still operates only at the symbol level and does not capture sequencing, layout, or contextual dependencies between tokens. Stitching together predictions for entire equations would require an additional detection and parsing layer, significantly complicating the pipeline. These limitations, combined with the need to handle full expressions rather than isolated glyphs, motivated our transition to a sequence-to-sequence equation model based on Pix2Tex.

Fine-tuned Pix2Tex Equation Pipeline Results:

Introduction

After validating the feasibility of symbol-level recognition, we shifted our equation pipeline to a sequence-to-sequence approach using Pix2Tex. Instead of classifying individual cropped symbols, Pix2Tex directly maps an input image of a handwritten equation to a LaTeX token sequence. The goal of this stage was to overcome the structural limitations of the CNN classifier and evaluate whether a transformer-based encoder–decoder model, fine-tuned on relevant datasets, could robustly handle real handwritten equations with fractions, roots, subscripts, superscripts, and multi-symbol expressions etc.

Data

Since the original Pix2Tex was trained on purely computer-generated images (im2latex-100k) of mathematical equations and their corresponding LaTeX source code, we need to fine-tune it on a handwritten dataset, CROHME, which provides a diverse collection of equation images with ground-truth LaTeX. However, CROHME consists of clean digital ink on uniform backgrounds (tablet-style handwritten formulas), while our front end receives photos of handwritten formulas on paper with shadows, noise, and perspective distortion. To reduce this domain gap, we preprocess input photos to make them more closely resemble CROHME-style images before being passed to Pix2Tex.

Performance

We run 10 epochs and store the weights for each of them. In particular, the 10th epoch achieves a validation accuracy of 0.5752. Compared to the modified equation pipeline, the lower validation accuracy here is reasonable because our task is now harder and the metric is sequence-level. Compared to the off-the-shelf Pix2Tex checkpoint, the fine-tuned model shows a clear improvement in both token-level accuracy and sequence-level correctness on handwritten equations. Although we sometimes observe symbol substitutions, we can better handle nested fractions and exponents, and become more consistent in the use of LaTeX structures such as *\frac*, subscripts, and superscripts. Furthermore, even though early epochs sometimes achieved slightly higher token-accuracy metrics during training, their predictions were less stable, with more syntactic errors and fragmented expressions. The checkpoint from the final epoch (e.g., epoch 10) produces a more coherent, syntactically valid LaTeX overall, and is therefore adopted as the default for our equation pipeline.

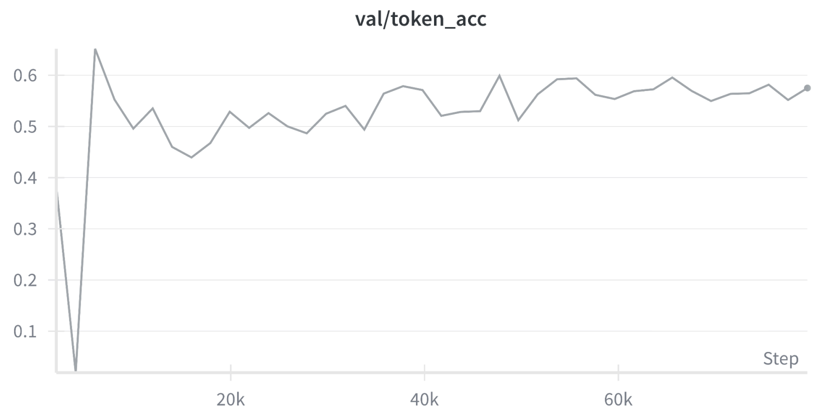


Figure 2: Validation accuracy

Takeaway

Fine-tuning Pix2Tex substantially improves the equation pipeline beyond what is achievable with symbol-only CNN classifiers. The model is able to integrate visual cues and spatial relationships directly into the generated LaTeX sequence, eliminating the need for separate symbol detection and manual assembly of expressions. While occasional errors remain, especially in highly cluttered or very stylized handwriting, the overall quality and usability of the LaTeX output are significantly better than both the baseline classifier and the off-the-shelf Pix2Tex model. This confirms that a sequence-to-sequence transformer, adapted to our target data, is a more scalable and maintainable solution for handwritten equation recognition within Texify.

Matrix Pipeline Results:

Introduction

The matrix pipeline is designed to convert handwritten numerical matrices into clean, structured LaTeX code. Unlike equations or text, handwritten matrices require robust cell segmentation, digit recognition, and structural reconstruction. Our approach leverages contour-based cell detection, padded bounding box, and digit-only OCR using Microsoft TrOCR (base-handwritten). This pipeline prioritizes segmentation accuracy and minimizes OCR ambiguity by explicitly restricting the vocabulary to numerical characters.

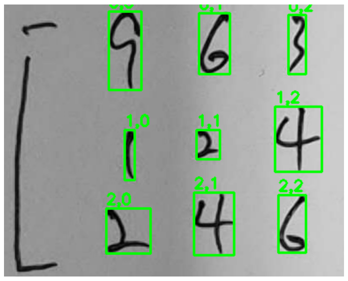


Figure 3: Example of matrix cell segmentation

Performance

Evaluation was performed on a collection of handwritten matrices captured using both an iPhone camera input and a standard webcam. The dataset includes:

- Single-digit and multi-digit entries
- 2x2, 3x3, and irregularly space matrices
- Variations in stroke thickness, lighting, and writing angle

We evaluated three recognition strategies:

1. MNIST
   MNIST-trained digit classifiers performed poorly. The model expected centered, symmetric, thin-stroke digits, causing heavy misclassification, near-zero robustness to handwriting variation, and confusion with angled or thicker strokes, making it unusable for real handwritten matrices.
2. EMNIST
   EMNIST expanded the alphabet but still failed to generalize to our handwritten digits. It's sensitive to stroke curvature, spacing, and crop tightness. As a result, it's also not suitable for production
3. TrOCR (Microsoft, base-handwritten)
   This model provided by far the best accuracy on real handwritten cell crops, and it benefits significantly from padded crops, high-resolution input, and stable cell segmentation. To eliminate hallucinated letters and punctuation, we constrained the decoder to output only numeric tokens (0-9), which resulted in higher token-level accuracy, elimination of non-numeric artifacts, improved consistency on mutli-digit entries, and more stable results across lighting and handwriting styles. In cases where segmentation was clean, TrOCR achieved very high digit correctness across all matrix cell positions. Interestingly, base-handwritten generated better results than base-stage1 in this case.

Takeaway

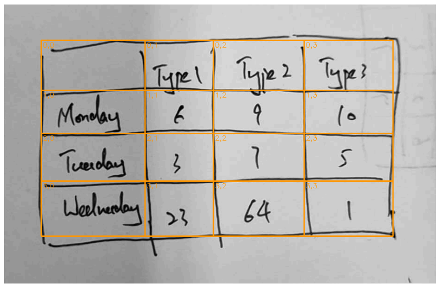
The contour-based segmentation + padded crop + digit-restricted TrOCR architecture forms a reliable and practical pipeline for handwritten matrix recognition. Traditional digit classifiers

(MNIST/EMNIST) do not generalize to real-world handwriting, while TrOCR's transformer-based OCR provides robustness to noise, angle variations, and non-uniform cell sizes. This pipeline successfully transforms handwritten matrices into accurate LaTeX representations and serves as a strong foundation for extending the system to larger matrices, mixed numeric-symbol matrices, or integration with end-to-end equation parsing.

Table Pipeline Results:

Introduction

The table pipeline is designed to convert handwritten tables into structured LaTeX tabular environments. Compared to equation or matrix parsing, table extraction is more complex because it must identify rows, columns, merged cells, cell boundaries, and multi-character handwritten entries. Our approach combines explicit line-structure detection with TrOCR-based cell-level OCR to produce accurate table reconstructions. The pipeline focuses on robustness to non-uniform handwriting, uneven cell borders, and imperfect scanning conditions.



	Type 1	Type 2	Type 3
Monday	6	9	10
Tuesday	3	7	5
Wednesday	23	64	1

Figure 3: Example of table grid segmentation

Data

Evaluation was conducted on a set of handwritten tables captured using mobile devices and a webcam. The dataset includes tables ranging from 2x2 to 6x6, both fully bordered and partially bordered table structures. Tables contain numeric entries, short handwritten words, multi-character strings, and empty cells.

Performance

The pipeline's performance can be divided into 2 components: structural detection and cell OCR:

1. Structural Detection
   The pipeline relies on fine-tuned vertical and horizontal line detectors, which identify long continuous strokes that correspond to row and column boundaries while filtering out

short or noisy line fragments. Debug visualizations confirmed that even imperfect, hand-drawn tables were reconstructed into consistent and stable grid structures.

2. Cell OCR
   TrOCR base-stage1 significantly outperformed lightweight character models. It's capable of reliably handling multi-character handwritten entries, and it is able to read short words, multi-digit numbers, symbolic shorthand, and slightly messy handwriting. OCR accuracy was noticeably influenced by segmentation quality; clean boundaries produced strong results.

Takeaway

The table pipeline effectively reconstructs handwritten tables into LaTex by combining reliable grid segmentation through horizontal and vertical line detection and TrOCR-based cell transcription. This hybrid approach enables strong performance on real handwriting, providing clean tabular environments that preserve the table structure - including spans and multi-line entries. The results demonstrate that a structure-first pipeline paired with transformer OCR offers a practical solution for handwritten table digitization, outperforming simpler grid or classical OCR techniques.

6. Timeline, Milestones, Roles

Timeline

- **11/1 - 12/1:** Iterative development and refinement - improve accuracy through data augmentation and error analysis.
- **12/1 - 12/5:** Expand the pipeline to table and matrix recognition tasks. Switch from training CNN's to fine-tuning Pix2Tex for the equation pipeline.
- **12/5 - 12/7:** Final tuning and final report preparation - consolidate results and visualizations.

Milestones

- Midterm Report (10/31): Working symbol recognizer + preliminary layout detection
- Final Report (12/11): Fully integrated system with evaluation results.

Duties

- Zihan: content classifier, matrix pipeline, table pipeline, frontend
- Shukai: equation pipeline, testing, report

7. Future Work

Several extensions could significantly improve Texify's performance and usability. A major direction is enhancing structural parsing for complex formulas. While the current Pix2Tex approach handles many cases, it can struggle when structures or symbols become nested and complicated, such as integrals or the expectation of a formula involving fraction and summation. Incorporating layout-aware models or explicit spatial parsing could improve accuracy for such structures. Another opportunity lies in expanding the fine-tuned dataset to include more images that are actually math formulas written on paper.

Beyond equations, the matrix and table pipelines also present clear opportunities for improvement. For matrices, future work could incorporate more robust cell-boundary detection using transformer-based segmentation models rather than relying solely on contour extraction. This could help address failure cases involving irregular handwriting, slanted entries, or uneven spacing. Additionally, replacing the current digit-restricted TrOCR with a domain-specific fine-tuned version may improve recognition of multi-digit or mixed symbolic entries. For tables, a more advanced structural parser—such as a graph-based layout analyzer—could improve merged-cell detection, border inference, and row/column alignment, particularly for partially drawn or unstructured tables. Incorporating a learned table-structure model would also help Texify generalize to more complex tabular formats encountered in notes, textbooks, or exams.

An especially promising extension is detecting equations and complex mathematical expressions inside matrices or tables, rather than limiting recognition to plain numeric or textual entries. Adding this capability would allow Texify to handle more realistic classroom and research notes, where mixed-content tables (e.g., matrices with symbolic entries, tabular derivations, annotated formulas) frequently occur.

Beyond enhancements to existing modules, Texify could be expanded into a multi-pipeline architecture that unifies several domain-specific recognizers. In addition to equations, matrices, and tables, future pipelines could include pure text recognition, symbol-level recognition, or mixed-layout pages combining text, diagrams, and math expressions. A unified dispatcher capable of routing each region to the appropriate specialized pipeline would greatly increase Texify's versatility and overall recognition accuracy.

On the frontend side, future development could focus on improving the user experience and reducing friction during input. Adding real-time feedback during cropping, live OCR previews, and automatic deskewing would help users obtain higher-quality results without manual intervention. Expanding support for additional input modalities—such as continuous webcam streaming for step-by-step capture, drag-adjustable bounding boxes, and stylus-friendly upload interfaces—would make Texify more versatile across devices. A dedicated results editor, where users can directly modify the generated LaTeX or highlight mistaken regions for reprocessing, would further enhance usability and help users efficiently refine their outputs.

8. Lessons Learned

Developing Texify highlighted several important lessons about mathematical OCR. First, structured content such as tables and matrices cannot be handled reliably by a single end-to-end model. Separating the system into focused pipelines proved far more dependable and easier to maintain. Second, the project reinforced how crucial diverse data is for handwritten recognition. Early models trained only on symbols or printed formulas failed to generalize, and fine-tuning on handwritten datasets was necessary for usable performance. We also learned that early training metrics like token-level accuracy do not necessarily reflect real-world quality. Later epochs, even with slightly lower token accuracy, often produced more coherent and reliable LaTeX outputs.

Finally, the project demonstrated the value of clear division of work and iterative integration. By developing equation OCR, table segmentation, and matrix processing in parallel and combining them gradually, our team avoided system-wide bottlenecks and improved each component more effectively. Overall, building Texify emphasized the importance of modular design, careful data selection, and structured experimentation when working with multimodal models.