Paper2Proof: Offline Full Page Latex Conversion

Bhaumik Mehta
University of Washington
bm87@uw.edu

Archit Kumar
University of Washington
akumar57@uw.edu

1 Introduction

Students and professors in technical fields frequently need to convert handwritten or printed mathematics into LaTeX. Whether transcribing a theorem from a research paper, digitizing homework solutions, or turning rough notebook sketches into professional documents, users often find that LaTeX, although powerful, remains time-consuming and tedious to type. Harkness and Watson report that the most time-consuming aspect of LaTeX-based courses is the formatting of mathematical expressions themselves [1], a pain point we experienced personally in our own coursework. This motivates exploring automated pipelines that can convert images of mathematics into executable LaTeX.

Traditional LaTeX OCR pipelines would include the following steps:

1. Conduct some preprocessing to clean the image
2. Segment characters through a segmentation method
3. Analyze the symbol layout by detecting structures (like superscript or radicals) and parsing them through some rule-based grammar or tree structure
4. Convert the representation back to LaTeX.

However, these systems were not at all robust and would fail when symbols touched or overlapped.

Historically, LaTeX OCR has been built around classical computer vision pipelines. These methods typically involve the following steps:

1. denoising and binarizing the input
2. segmenting characters via connected components or projection profiles
3. inferring structural relationships such as superscripts, radicals, and fraction bars using hand-engineered grammars
4. reconstructing LaTeX from a symbolic parse tree

While elegant in theory, these systems are notoriously fragile: overlapping strokes, handwriting irregularities, or touching symbols frequently break the segmentation and produce cascading structural errors. Traditional math OCR systems like InftyReader and various academic prototypes struggle especially with the very things that make handwritten mathematics difficult—variable spacing, inconsistent baselines, irregular symbol shapes, and non-linear layouts.

Deep learning methods represent the modern direction for mathematical expression recognition. Sequence-to-sequence architectures such as Im2LaTeX introduced encoder–decoder models capable of mapping rendered mathematical images to LaTeX without an explicit symbolic grammar. More recently, Vision Transformers (ViTs) have dramatically improved visual understanding, enabling models like pix2tex (LaTeX-OCR) to achieve strong performance on single, clean mathematical formulas. However, these models are generally trained on large synthetic datasets—such as Im2LaTeX-100k—containing high-quality rendered equations rather than handwritten notes. They

Preprint. Under review.

also assume that the input contains exactly one formula of near-standard size, making them poorly suited for full-page documents.

Large multimodal LLMs offer another path forward. Commercial systems such as MathPix or GPT-4o mini can now parse entire pages of handwritten or printed math with impressive fidelity. Yet, they come with drawbacks: they are compute-heavy, require online inference, have unpredictable latency, and are not easily deployable in offline or low-resource settings. Their performance is also tightly bound to proprietary training data. For a lightweight, offline project, these models—while powerful—are impractical or inaccessible.

This creates a clear gap: deep learning models that perform well on single equations exist and are open-source, but they cannot operate on full pages. Conversely, end-to-end models that handle full-page understanding are too large to run locally. Our work aims to bridge this gap. Rather than designing a new LaTeX OCR model, we extend pix2tex to full pages by developing a classical computer-vision pipeline that isolates equations from handwritten or online documents. Our key insight is that segmentation allows existing lightweight models to operate on full documents without retraining. The rest of this paper describes our CV pipeline, evaluates its segmentation quality and downstream recognition accuracy, and highlights both strengths and challenging failure cases.

In terms of contributions, Archit handled the process of writing the pipeline that would process an image and use the model. He then created a frontend that packaged our solution into an easy to use website. Bhaumik handled stress testing of the system, finding failure cases and providing recommendations. Then he created the write up.

2 Method

Before discussing our improvements, we provide more details about the model. The problem pix2tex aimed to solve was taking an image of one math formula or equation and outputting the corresponding LaTeX code. It used a Vision Transformer (ViT) encoder with a ResNet backbone to encode an input image. Then a Transformer decoder would output the sequence of predicted LaTeX tokens. Since the training set mostly consisted of images of a certain size, the algorithm would first try to automatically resize input images to best resemble the training data.

The model used Im2LaTeX-100k along with publicly available LaTeX formulas from sources like Wikipedia and arXiv as training data. They would also augment the training set by rendering each formula with various math fonts using XeLaTeX and converting the rendered PDF back into the PNG format. The end-to-end algorithm's performance is summarized in Table 1.

Model	BLEU Score	Edit Distance	Token Accuracy
pix2tex (LaTeX-OCR)	0.88	0.10 (normalized)	0.60

Table 1: Reported performance metrics for the pix2tex (LaTeX-OCR) model

The BLEU score describes how many n-grams from the model's output appear in the reference LaTeX, examining the quality of local translations. The edit distance measure the minimum percentage of character edits required to convert model output to the ground truth, revealing the correctness of the solution. The accuracy measures the next-token prediction performance. The metrics show that pix2tex can reasonably convert single line equations. It is able to produce sensible local output and capture small parts of the equation well (high BLEU). However, some human editing is probably still needed to produce the final latex (Edit distance only moderately low). The lower token accuracy isn't too concerning given that there are also multiple LaTeX formulas that correspond to one image.

We decided to utilize computer vision algorithms to augment the algorithm and create a system that could tackle the problem of translating of full pages of LaTeX. Our key insight into extending the capability of the model was to extract individual equations from the document, apply the model to each equation, and ensemble the LaTeX back together. The details of this pipeline are summarized in Algorithm 1 below.

Algorithm 1: Pipeline for Page-to-LaTeX Conversion

```
1: Input: Raw document image
2: Output: Reconstructed LaTeX document
```

```
3:  procedure ProcessImage
4:      Step 1: Correct Skew
5:      Convert image to grayscale
6:      Detect edges using thresholding and Canny edge detection
7:      Apply Hough line detection to extract dominant lines
8:      Filter out near-vertical lines and compute the median line angle
9:      Rotate the entire image by this estimated angle
10:     Step 2: Clean and Binarize Image
11:     Convert to grayscale
12:     Apply Gaussian blur to smooth noise
13:     Perform adaptive thresholding (block size = 25) to binarize image
14:     Remove small specks, pixels, and tiny artifacts via morphological opening (shrink white
        regions/delete lone white pixels followed by dilating the image back)
15:     Step 3: Segment Equation Regions
16:     Perform horizontal dilation to ensure adjacent characters touch
17:     Identify connected components and form bounding boxes (with padding = 3)
18:     Merge any bounding boxes that overlap
19:     Step 4: Equation Recognition
20:     for each segmentation box do
21:         Run pix2tex model to generate the corresponding LaTeX code
22:     end for
23:     Step 5: Document Reconstruction
24:     Combine all generated LaTeX expressions
25:     Wrap with appropriate LaTeX headers, footers, and formatting
26: end procedure
```

Design Choices and Rejected Alternatives

Several alternative design decisions were considered but discarded during development. For instance, we experimented with 2D morphological dilation rather than purely horizontal dilation. Although this improved connectivity for handwritten superscripts, it frequently merged adjacent equation lines, degrading segmentation precision. We also considered using global thresholding for binarization, but uneven illumination in handwritten pages led to severe loss of structure, motivating our adoption of adaptive thresholding. In skew correction, PCA-based angle estimation and projection-profile methods were tested but proved sensitive to handwritten curvature and noise; the median Hough-line angle was substantially more stable. Finally, we evaluated hierarchical clustering of bounding boxes as an alternative to overlap-based merging. While appealing in theory, clustering introduced sensitivity to hyperparameters and produced inconsistent grouping across handwriting styles.

That being said, the current steps each serve important roles. The Skew processing module is included to account for irregular/rotated images, which the model does not handle by default. Heavily inspired by the line detection assignments, we use the same algorithm but use the detected lines differently. Under the assumption that the writing would be globally tiled the same way (if the picture is taken at angle this is true. Also, if people write with a slant the slant is usually consistent) we can rotate by a representative angle which we choose to be the median. We also include filtering steps and thresholding while deciding on the angle to rotate by (for example, lines that are near vertical are filtered out since those typically result from artifacts like page borders instead of equations).

In order to clean the image, we apply a variety of tricks learned from class. For instance, we use gaussian blurring followed by localized thresholding to generate masks and retain only important equation pixels. We also use thresholding to get rid of disturbances.

The next step involves the segmentation process. The idea here is that equations all lie in one line, so we can dilate characters horizontally to make characters in an equation touch. We can then use connected regions to identify components. This key idea is expanded on in step 3. For example, we also make sure to merge bounding boxes since structures like superscripts may lead to adjacent characters not touching even after dilation despite belonging to the same equation. The fix is that even in this case, the bounding box would still overlap so a post step of merging bounding boxes takes care of that edge case.

Finally, we are able to apply the model to each of the individual equations we have detected and reconstruct the LaTeX in steps 4 and 5. Our web app also further includes a step to allow the user to attempt to regenerate the LaTeX for any equations that the model failed to produce valid LaTeX for. Since the model is non-deterministic, this addition is able to sometimes fix incorrect generations.

3 Experiments

In this section we examine some quantitative metrics and qualitative examples to determine the performance of the model and identify areas of improvement. Since our main insight involved using algorithms to identify equations, we discuss this portion of the algorithm in the first subsection. In the following subsection we examine the performance of the pipeline in full.

3.1 Image processing pipeline

We begin by examining how well our proposed image processing pipeline performs. We begin by defining the following metrics:

$$\text{Recall} = \% \text{ of true equations detected} \tag{1}$$
$$\text{Precision} = \% \text{ of detected equations which are actual equations} \tag{2}$$
$$\text{Segmentation Rate} = \text{\# Predicted Equations / \# Ground Truth Equations} \tag{3}$$

We crafted a dataset of $n = 20$ images by randomly sampling from willing participants' handwritten LaTeX samples and the Im2LaTeX-100k test set. Our expectation was that we would be able to deal with rotations and in-line equations well. Samples that were pulled from online would also be easier to deal with since our segmentation dilation was tuned to work for the vertical and horizontal spacing seen in those examples. This also meant that we expected worse performance on handwritten images. Equations that were messily written, overlapping, or spanning multiple lines would also fail our algorithm. Finally, our assumption that equations would all be written with the same tilt held true throughout all our samples.

The quantitative results of our processing algorithm are summarized in Table 2. The high recall makes sense consider examples like that in figure 2. Equations are always captured by the segmentation so always included in some equation bounding box. On the other hand, the equation boxes easily mistook tricky cases like the braces in figure 2 or other handwritten artifacts. Still, on online images, the algorithm performed well as discussed with figure 1. The segmentation rate being smaller than 1 also makes sense as our algorithm would lean towards grouping true equations together over simply missing them.

Recall	**Precision**	**Segmentation Rate**
1.0	0.7	0.82

Table 2: Performance metrics for the image processing step

These quantitative findings highlight two key behaviors of the segmentation stage. First, the perfect recall indicates that our dilation-based strategy strongly favors over-inclusion of equations even when the page contains noise or small artifacts. Second, the precision of 0.7 reflects a nontrivial number of false positives, typically arising from handwritten artifacts, long braces, or lighting-induced edges that visually connect separate expressions. This trade-off is intentional, since downstream LaTeX OCR is much easier to apply when every true equation is captured at least once, even if some extra regions must be discarded. To better understand how these effects manifest in practice, we next examine representative qualitative examples.

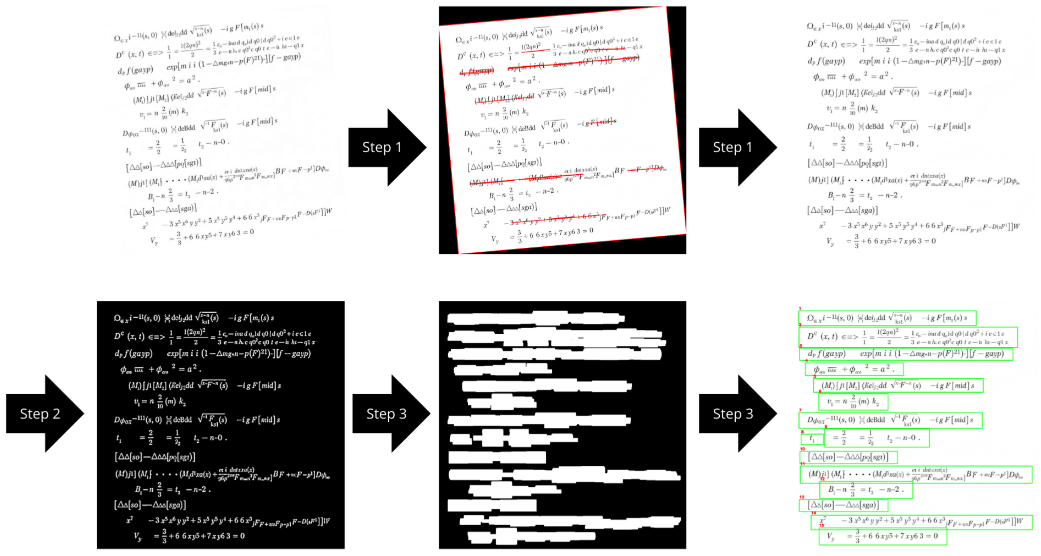

Figure 1: Image processing pipeline on a successful case. The first two transformations entail the skew correction process (Step 1). The next transformation shows the result after cleaning and binarization (Step 2). The following two transformations reveal the segmentation and resulting equation recognition (Step 3).

A qualitative analysis of our performance was extremely revealing. To start, figure 1 shows an example where our algorithm performed well. It demonstrates that the algorithm is able to adjust for skew, capture well-formatted, single-line equations, and accurately create bounding boxes. This makes sense as our algorithm was specifically tuned to such examples from the Im2LaTeX-100k dataset. On the other hand, 2 demonstrates an insightful failure case. At the start, lines are not detected due to the relative large size of each character leading to much horizontal whitespace within the equations. Afterwards, binarization picks up on the edges of the uneven lighting, which we didn't account for. To make matters worse, the braces written by the user add connections between the segmentation of each of the equations, making the algorithm mistakenly believe there is one large equation. This failure case violates various implicit assumptions we'd made and reveals challenging cases for each step of the algorithm-based approach to preprocessing.

Taken together, these two examples span the spectrum of scenarios our image processing pipeline is designed to handle. The clean, synthetic-style page in figure 1 closely matches the distribution of the Im2LaTeX-100k dataset and showcases the best-case behavior of skew correction, binarization, and horizontal dilation. In contrast, the handwritten page with uneven illumination and connecting braces in figure 2 exposes how small violations of our assumptions can accumulate across stages. Presenting these side by side helps clarify which phenomena are fundamentally hard for the classical CV pipeline and which are primarily artifacts of particular writing styles or capture conditions.

3.2 Model performance with pre-processing

Now that we understand the effect the page pre-processing step, we examine how the entire pipeline performs. We begin by defining the following additional metric:

$$\text{Exact Match Rate} = \%\text{ of equations that were exactly correct} \tag{4}$$

We use the same hand-curated dataset described in the earlier section. Note that the base model by itself fails and produces LaTeX that doesn't render for every example in our dataset. The quantitative result of our experiment is described in Table 3. The normalized edit distance rises by such a large amount due to the outlier failure cases in the preprocessing step. When multiple equations are

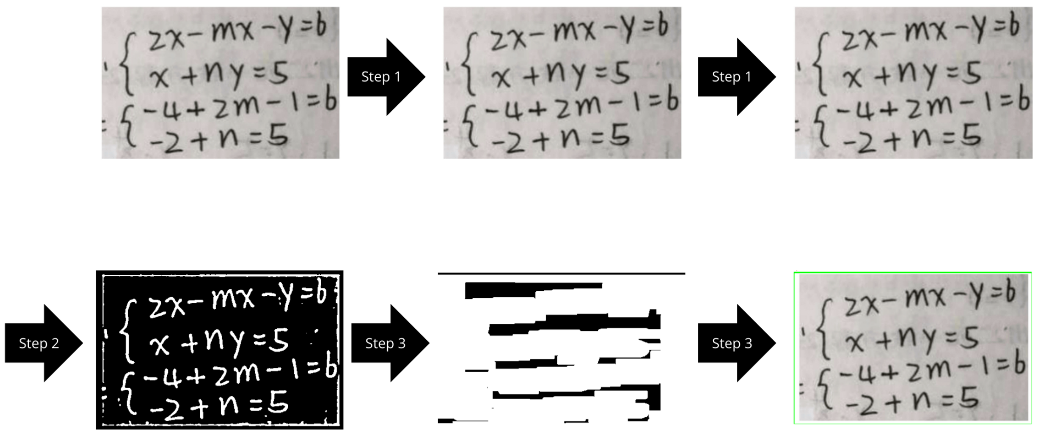

Figure 2: Image processing pipeline on a failure case. The steps correspond to the same transformation as in Figure 1

combined into the same image, the model struggles and collapses, produces a largely unusable string. Still, most images are parsed into individual images properly and retain the same edit distance of around 0.1. The exact match rate is also much lower because despite our processing step correctly identifying equations, the base model simply struggles to perfectly classify individual equations. This being said, just as mentioned before, the lower edit distance in those cases makes it simple to fix small mistakes and generate the exact latex needed.

Edit Distance	**Exact Match Rate**
0.24 (normalized)	0.15

Table 3: Performance metrics for our end-to-end pipeline

These end-to-end results underscore the interaction between our preprocessing pipeline and the underlying pix2tex model. The increase in normalized edit distance is driven by a small number of outlier cases in which segmentation fails, causing several equations to be merged into a single image that the model was never designed to handle. In contrast, whenever the preprocessing step produces clean, single-equation crops, the edit distance remains close to the originally reported value of 0.1, indicating that the OCR model behaves as expected. The relatively low exact match rate therefore reflects fundamental limitations of the base model rather than deficiencies in the segmentation strategy. To further disentangle these effects, we next turn to qualitative examples that isolate cases where preprocessing succeeds but recognition still introduces noticeable errors.

Now we take a qualitative look at our algorithm as well. Figure 3 demonstrates a success case. The processing step is able to oriented the LaTeX correctly and provide well-fitting bounding boxes for each individual equation. The model itself is then able to infer on the type of images it is intended to be used for. There are some characters which aren't handled properly by the model. For example, in the first equation the $j = 1$ seems to cause many issues with an additional underline appearing. The model also fails to write the overline over the $\sum$ character. Later equations also exhibit similar issues with a Φ being confused for a φ. The third equation is matched perfectly. These issues likely arise due to weird character combinations appearing somewhat out of distribution of the model's training data. Still, we would classify all equations as a success even though the first two were not exact matches, reinforcing the claim that although the exact match rate isn't perfect, the algorithm still performs well under alternative perspectives.

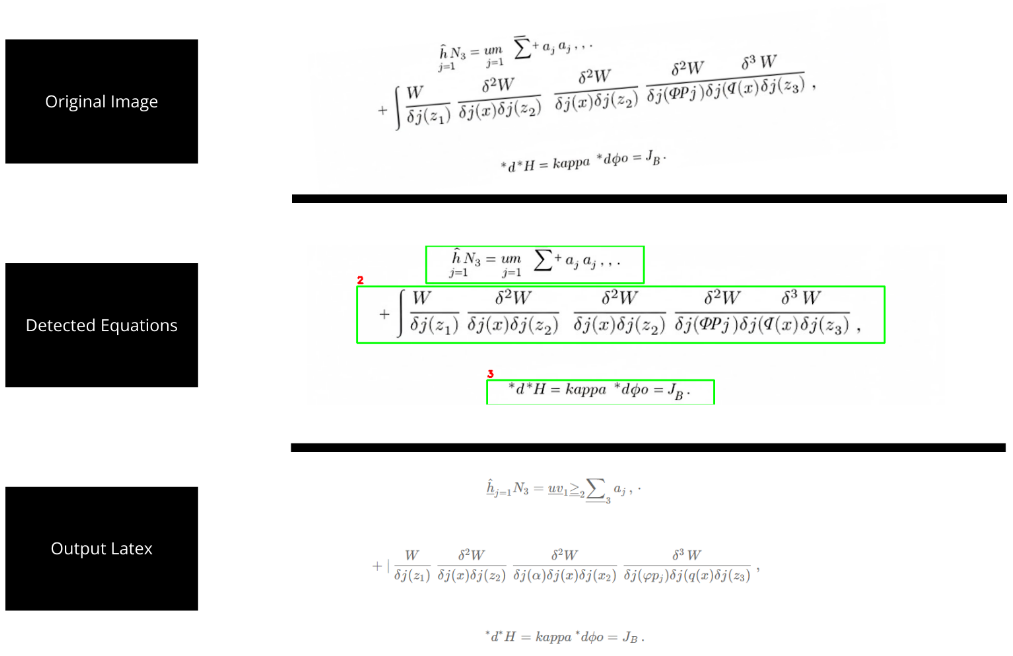

Figure 3: Image processing pipeline on a failure case. The steps correspond to the same transformation as in Figure 1

Figure 4 further reinforces our claims about the base model. The equations are identified correctly by our preprocessing algorithm. Then the model makes slight errors including swapping characters, mislabeling subscripts and superscripts, and confusing accents. Still, the edit distance remains quite low (as expected from the base model) and the generated LaTeX could be fixed with minimal effort.

Both examples demonstrate that our key idea to decompose documents into equations with computer vision has merit. The model is able to perform at a level similar to its original capability and any LaTeX errors are a product of known limitations of the model.

It is still interesting to look through other examples to gain insights into the model and algorithm. Figure 5 demonstrates two interesting unspoken limitations. For one, most of the training data only consists of Greek letters, so the model struggles to recognize digits! It instead attempts to find similar characters, representing 8 with a $\mathcal{M}$ or $\bar{S}$ and 0 with an O. Interestingly the number 10 seems to appear often enough that the model is able to correctly choose 10 frequently. If the image were to contain English text, the model would probably still attempt to coerce the characters into their nearest Greek counterpart.

The other limitation has to deal with arrays. The algorithm is unable to handle equations that lie on the same row. Currently, each equation is converted to LaTeX and then given its own equation environment. However, equations on the same line would require their own custom array environment to be displayed properly. Equations with roughly overlapping y coordinates could be placed in the same row into an auto generated array environment, however, such a hard coded algorithm wouldn't be able to handle horizontal spacing or other formatting quirks very well.

4 Conclusion

Converting full pages of mathematics into LaTeX efficiently and accurately remains a nontrivial challenge. Large multimodal models can address this problem but are impractical for offline or low-resource environments. Meanwhile, lightweight open-source models such as pix2tex excel at single-

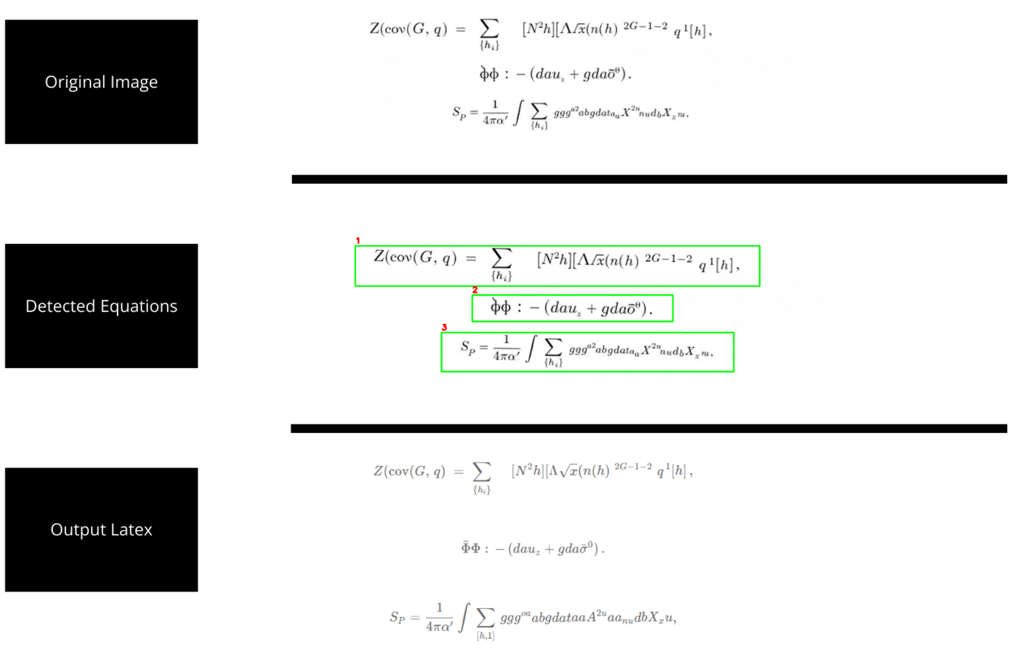

Figure 4: Image processing pipeline on a failure case. The steps correspond to the same transformation as in Figure 1

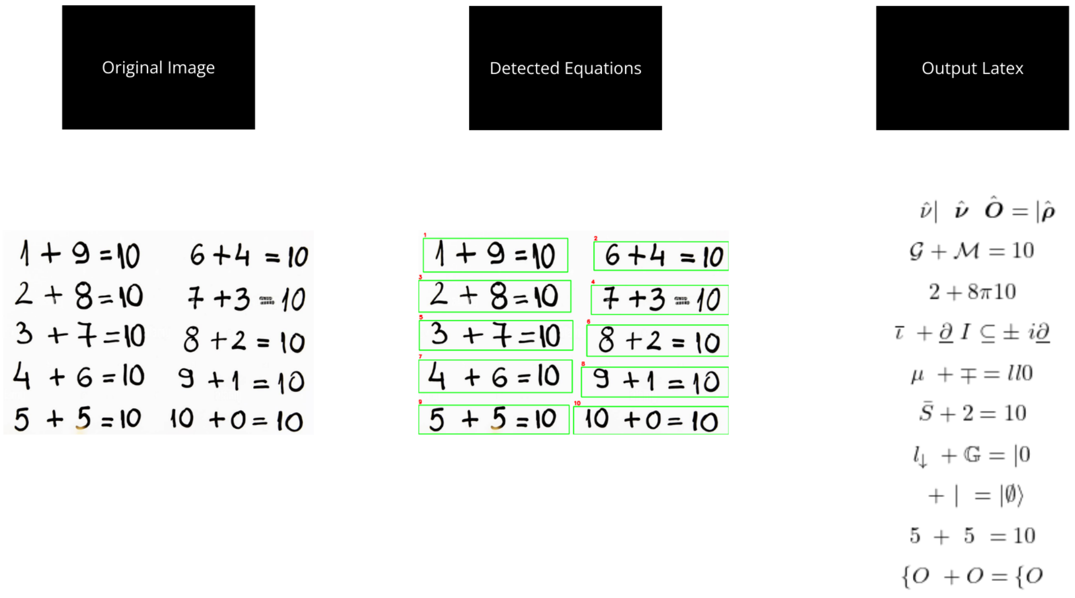

Figure 5: Image processing pipeline on a failure case. The steps correspond to the same transformation as in Figure 1

equation transcription but cannot directly process multi-equation pages. Our work demonstrates that classical computer vision techniques can serve as an effective bridge between these extremes.

By incorporating skew correction, adaptive thresholding, horizontal morphological dilation, connected-component analysis, and bounding-box refinement into an equation detecting algorithm, we find that we can reliably isolate individual mathematical expressions from a page. The high segmentation recall (Table 2) confirms that our approach almost never misses an equation when the underlying assumptions hold. The segmentation precision and failure cases reveal the natural trade-offs of morphological approaches, particularly in the presence of handwriting artifacts, uneven lighting, or unusual spacing patterns. Nevertheless, our method performs particularly well on online mathematical images such as those from Im2LaTeX-100k, where spacing and character structure are more predictable.

When paired with pix2tex, the system achieves recognition performance comparable to the model's original single-equation capability. The exact match rate remains modest (Table 3), reflecting known limitations of pix2tex—such as confusion between visually similar symbols (like Φ and φ), difficulties with accents, struggles with digits, and occasional rendering failures. Still, the low edit distance in successful cases reinforces our claim that the resulting LaTeX is typically only a few small corrections away from being fully accurate. Under a realistic user workflow, these small corrections are substantially less labor than typing entire formulas manually.

Our qualitative analysis also surfaced several deeper limitations that point toward future work. The model's strong bias toward Greek characters (Figure 5) highlights gaps in the training distribution. The inability to handle same-row equations reflects a broader limitation in understanding layout structure beyond simple line segmentation. More robust approaches to thresholding and lighting correction would mitigate failure cases like those in Figure 2, where uneven illumination and hand-drawn connecting braces break key assumptions of our method.

Future extensions could include more sophisticated page-layout modeling, such as detecting multi-line equation blocks, arrays, and piecewise functions. Larger and more diverse training sets which incorporate handwritten digits, mixed text-and-math regions, and varied writing styles—would help the underlying OCR model generalize better. Lightweight finetuning methods such as LoRA adapters [2] offer a promising path for enhancing pix2tex without requiring full retraining. Finally, integrating confidence estimates and feedback from rendered LaTeX could enable iterative self-correction loops or automated flagging of uncertain regions.

Overall, our results show that classical computer vision still has an important role to play in modern document understanding. By combining traditional segmentation with modern sequence-to-sequence recognition models, we offer a practical, offline approach for converting full pages of mathematics to LaTeX that is accessible, extensible, and effective across a wide variety of common use cases.

Code for this project can be found at https://github.com/ArcCreate/Paper2Proof/

References

[1] Ira Harkness and Justin Watson. Improving efficiency and consistency of student learning assessments: A new framework using latex. In *Proceedings of the 2024 ASEE Annual Conference & Exposition*, Portland, OR, 2024. American Society for Engineering Education.

[2] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.