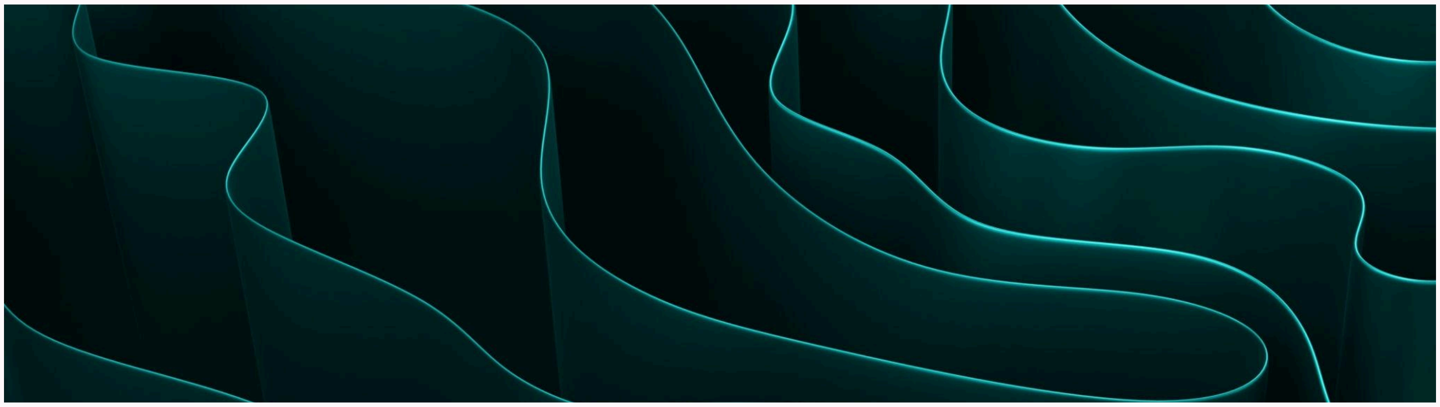


Technical Report

29-01-2026

Automated Error Bar Detection in Scientific Plots

Prepared by

Sabit Ahamed Preanto

BSc in Computer Science and Engineering
Daffodil International University
Phone: 01879133117
Email: preanto15-5059@diu.edu.bd

Submission Deliverables

- GitHub Repository: https://github.com/preanto/Automated-Error-Bar-Detection-in-Scientific-Plots/
- Generated Dataset: https://drive.google.com/drive/folders/1haFYteGqtVtG_m_YnNDHwJomvJ7SREGH?usp=sharing

01 ABSTRACT

This project was created to fulfill an engineering problem that deals with the automated detection of error bar endpoints in scientific plots, which is an essential part of the digital extraction process in scientific research. There are two main technical phases in this project: the generation of large-scale synthetic data, as well as the development of a high-precision detection pipeline. First, a Synthetic Dataset Generation pipeline was created, which generates scientific plot images with corresponding JSON annotations, totaling 3,000 images. These images are randomized in terms of line styles, colors, and markers, mimicking actual scientific plot images in the real world.

The second phase entailed creating the Error Bar Detection Pipeline. This pipeline takes an image and the coordinates of the data points as inputs. It then uses Vertical Intensity Profiling to traverse the vertical axis of each data point. This process involves using inverse binary thresholding to identify the exact pixel at which an error bar ends. This method is based on computer vision principles rather than machine learning to ensure accuracy and efficiency. The final results will comprise a technical report, the created dataset, and a GitHub repository for public use containing the entire execution logic.

02 INTRODUCTION

The extraction of data from scientific visualizations is an essential aspect of contemporary meta-analysis and the quest for research reproducibility. Although data points represent central tendencies, it is the error bars that represent standard deviation, standard error, or confidence intervals that are essential in understanding the statistical significance of the results. The extraction of data from legacy publications or images is traditionally not automated. Instead, it is done through manual methods that are both time-consuming and susceptible to human error. The purpose of this report is to outline an engineering solution that is dedicated to the automated extraction of the end points of the error bars. The solution is centered on developing a robust pipeline that links synthetic data generation with high-precision computer vision. The solution presented in this report is dedicated to providing a high level of methodological rationale and algorithmic robustness that is capable of overcoming the challenges of varied plot styles, noise, and anti-aliasing effects that are typical of scholarly visualizations.

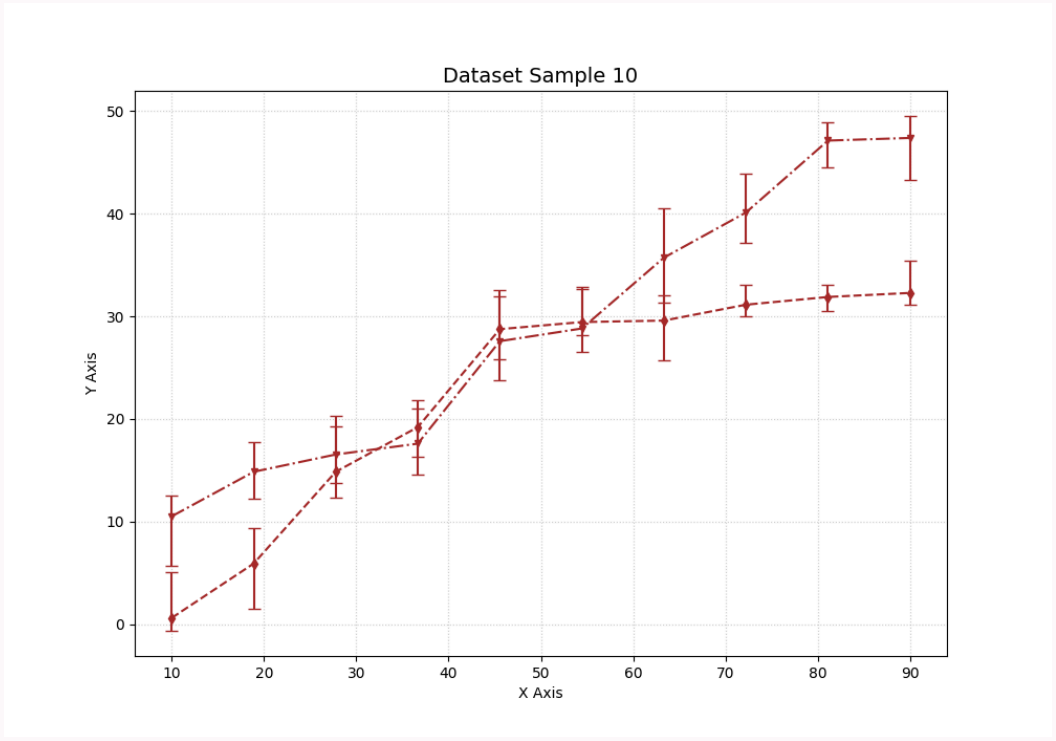
03 SYNTHETIC DATA GENERATION

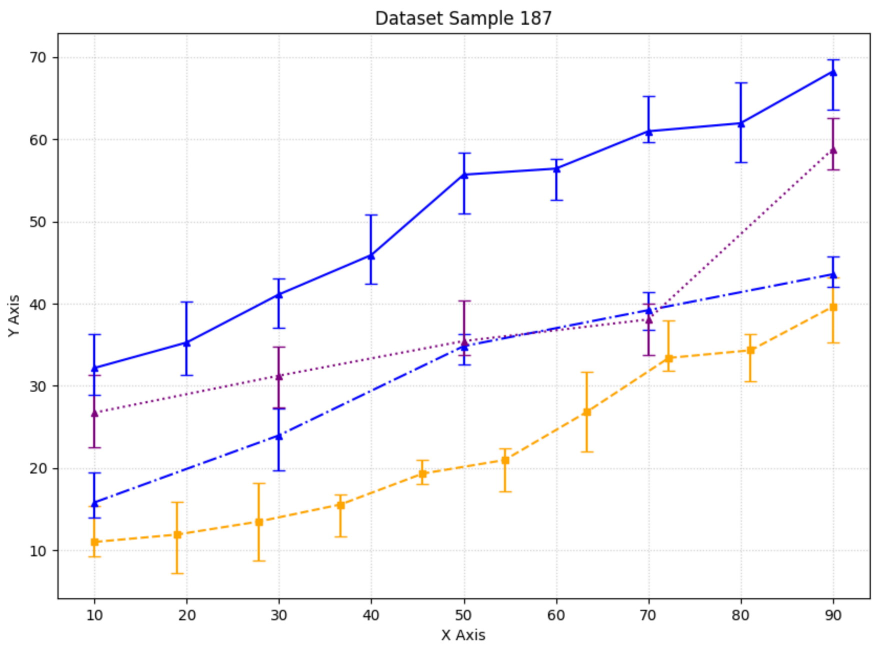
In order to create and validate the detection pipeline, a high-scale synthetic dataset consisting of **3,000** scientific images was generated using a randomized procedural model. Leveraging existing and well-established plotting libraries, the generation script was implemented to mimic the structural diversity that is typically observed within the pages of existing journals. This is accomplished through the use of different line types (solid, dash, dot), marker types (circle, square, diamond), and randomized color schemes. This approach is well-aligned with the methodologies that are currently discussed within the literature, such as the Journal of Big Data, where synthetic datasets are utilized to train and evaluate computer vision algorithms when the cost of human labeling is prohibitively expensive. In this way, the parameters of the generation model were used to create high-fidelity ground truth annotations, which were then saved in JSON format to capture the exact pixel coordinates of the data points, as well as the extent of the vertical error bars. This approach provides an objective quantitative evaluation of the detection algorithm, removing the variables of extreme image degradation, yet providing enough variability to evaluate the generalized robustness of the approach across multiple "groups" or data series within the same image.

Algorithm 1. Procedural Plot Generation

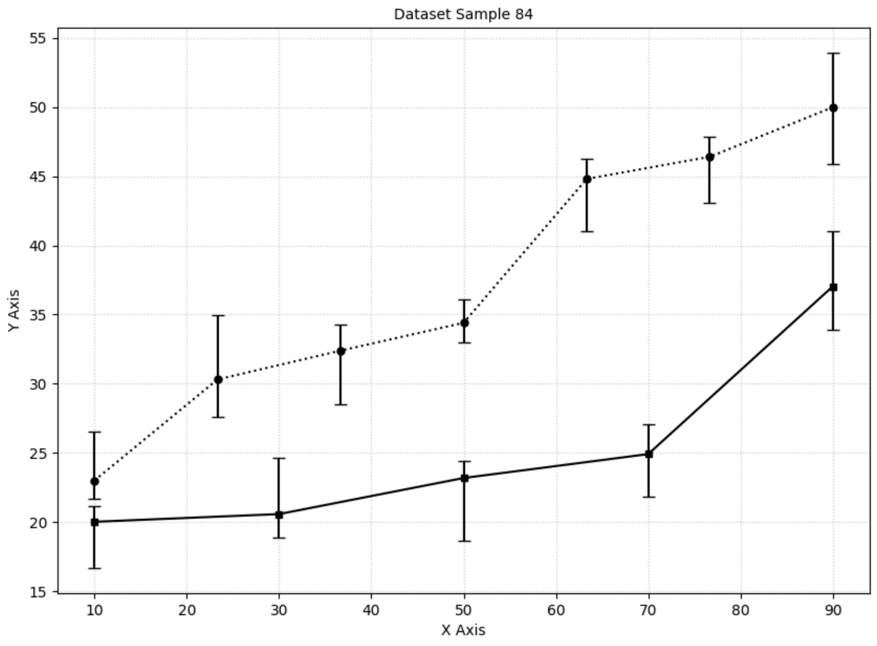
```
Input:
    Number of plots to generate
    Number of data points N per line
Output:
    Plot image (PNG)
    Annotation metadata (JSON)
1: Initialize canvas with fixed size and high DPI
2: Randomly select number of data series L ∈ [1, 5]
3: Randomize global aesthetics (fonts, grid, axis labels)
4: for each line Li where i = 1 to L do
5:     Randomly select line style, marker type, and hex color
6:     Generate N data points (x, y) using a randomized growth function
7:     Compute topBarPixelDistance ∈ [2%, 8%] of y-axis range
8:     Compute bottomBarPixelDistance ∈ [2%, 8%] of y-axis range
9: end for
10: Render plot using plotting library
11: Transform data coordinates to pixel coordinates
12: Invert y-axis to convert bottom-left origin to top-left origin
13: Save rendered plot as PNG
14: Save metadata as JSON containing image name and data points grouped by line
```

Some sample synthetic data:

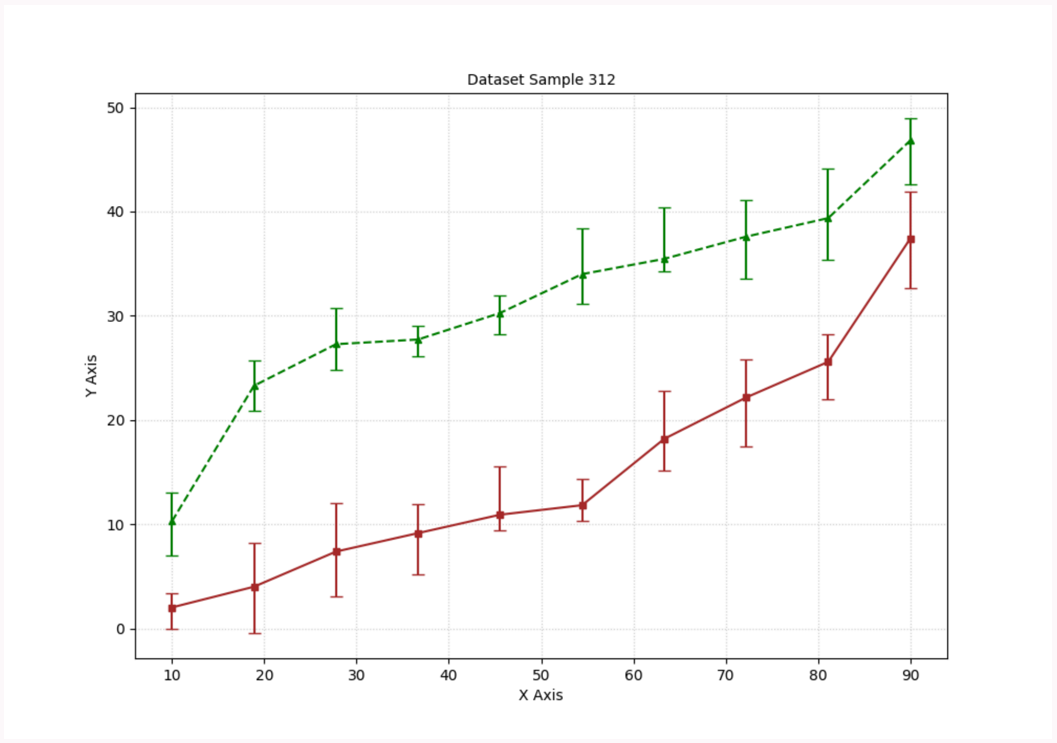




Dataset Sample 187
Y Axis
X Axis



Dataset Sample 84
Y Axis
X Axis

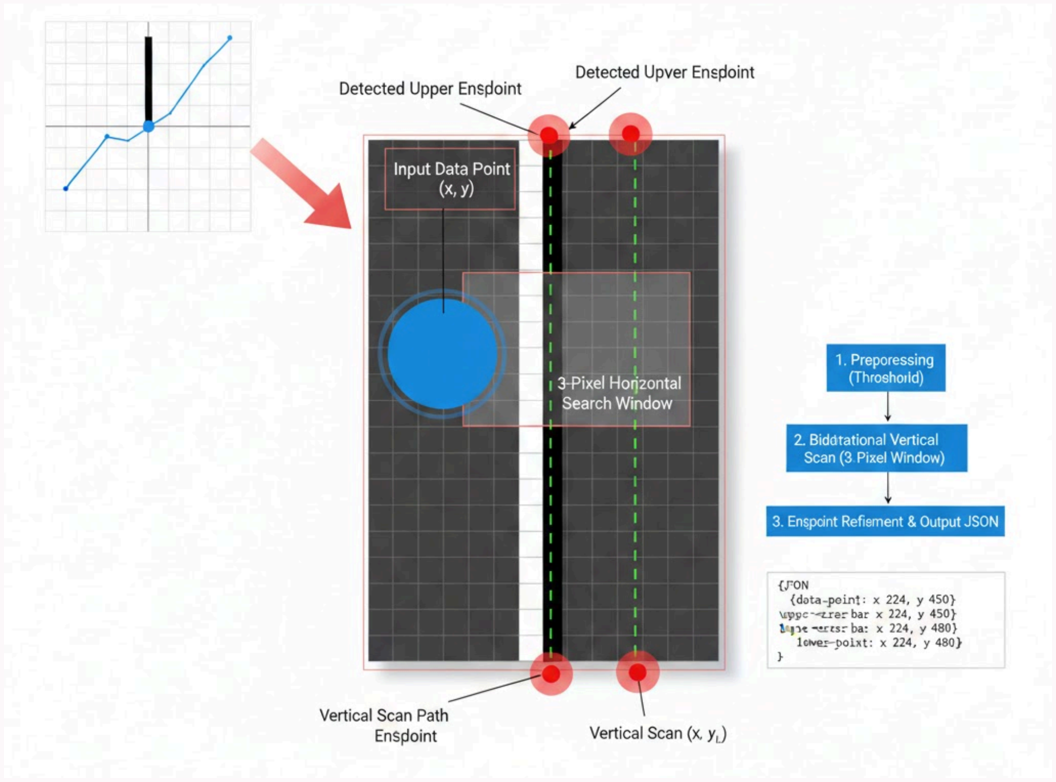


04

ERROR BAR DETECTION APPROACH

The fundamental detection logic uses an Intensity-Based Vertical Profiling algorithm, which is a traditional computer vision algorithm providing pixel-accurate results that go beyond the capabilities of most standard bounding-box detection models. The process starts with the preprocessing of the input image, converting it to a grayscale image and then using an inverse binary threshold, which essentially picks out the dark graphical components of the error bars as high-intensity "signal" against the black background. The algorithm begins with the given data point coordinates and uses a bidirectional vertical scan along the y-axis. This is similar to line-tracing algorithms that have been discussed in the International Journal of Computer Vision. In order to counter the effect of anti-aliasing, whereby a single line could extend over several pixels horizontally with varying intensity, the script uses a 3-pixel horizontal region check at every vertical step. This ensures that the detection process is not prematurely halted by a single "weak" pixel, allowing the algorithm to follow the error bar until a real gap in the graphical data is reached, which marks the true upper or lower boundary.

Figure 1. Error Bar Detection via Vertical Intensity Profilling


Detected Upper Enspoint
Detected Upver Enspoint
Input Data Point (x, y)
3-Pixel Horizontal Search Window
1. Preporessing (Throshoild)
2. Biddtational Vertical Scan (3 Pixel Window)
3. Enspoint Refiement & Output JSON
Vertical Scan Path Enspoint
Vertical Scan (x, y$_L$)

```
{JᐧON
  {data-peint: x 224, y 450}
\uppc-erer bar x 224, y 450}
lope -ertsr ba: x 224, y 480}
   lower-poixt: x 224, y 480}
}
```

Algorithm 2. Error Bar Detection Algorithm

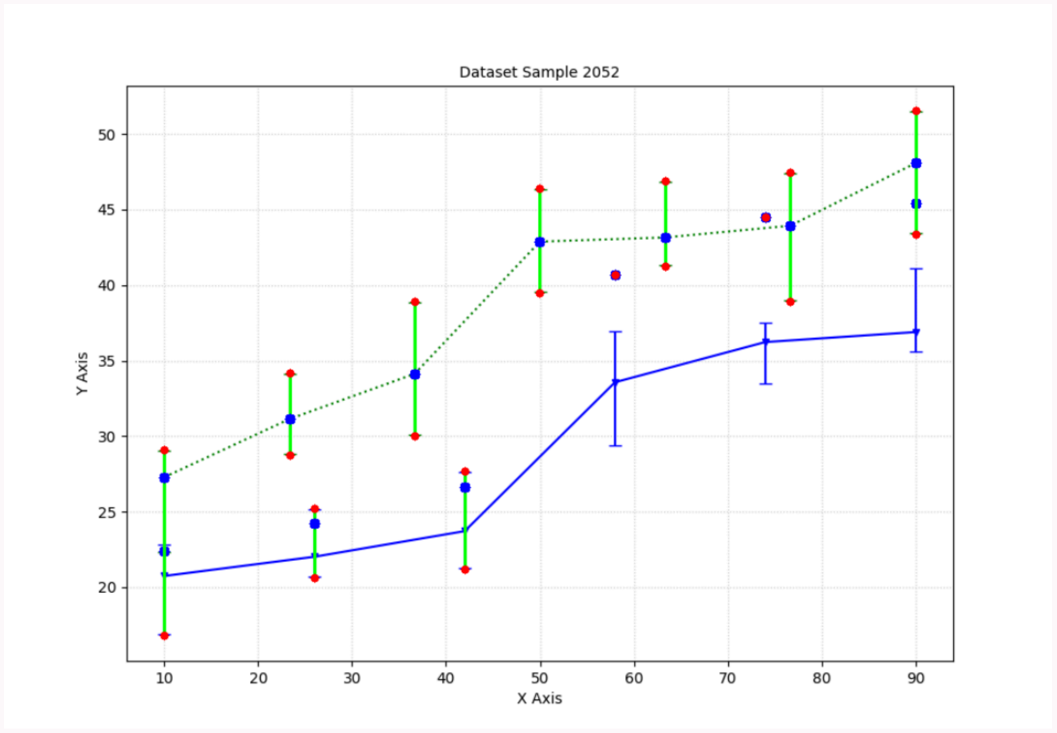
```
Input:
    Plot image (PNG)
    Data point anchors from JSON

Output:
    Error bar endpoints (JSON)

1: Convert input PNG to grayscale image G
2: Apply inverse binary threshold (T = 220) to G
3: Create binary mask B where:
       error bar pixels = 255
       background pixels = 0

4: for each data point P(x, y) in JSON do
5:     Round coordinates to nearest integer (ix, iy)

6:     // Upper error bar search
7:     y_up ← iy
8:     while y_up ≥ 0 do
9:         if no pixel with value 255 exists in
B[y_up][ix-1 : ix+1] then
10:            break
11:        y_up ← y_up - 1
12:    end while
13:    upper_error_bar ← last valid y_up

14:    // Lower error bar search
15:    y_down ← iy
16:    while y_down < image_height do
17:        if no pixel with value 255 exists in
B[y_down][ix-1 : ix+1] then
18:            break
19:        y_down ← y_down + 1
20:    end while
21:    lower_error_bar ← last valid y_down

22: end for
23: Store results in JSON format using original
floating-point (x, y) coordinates
```

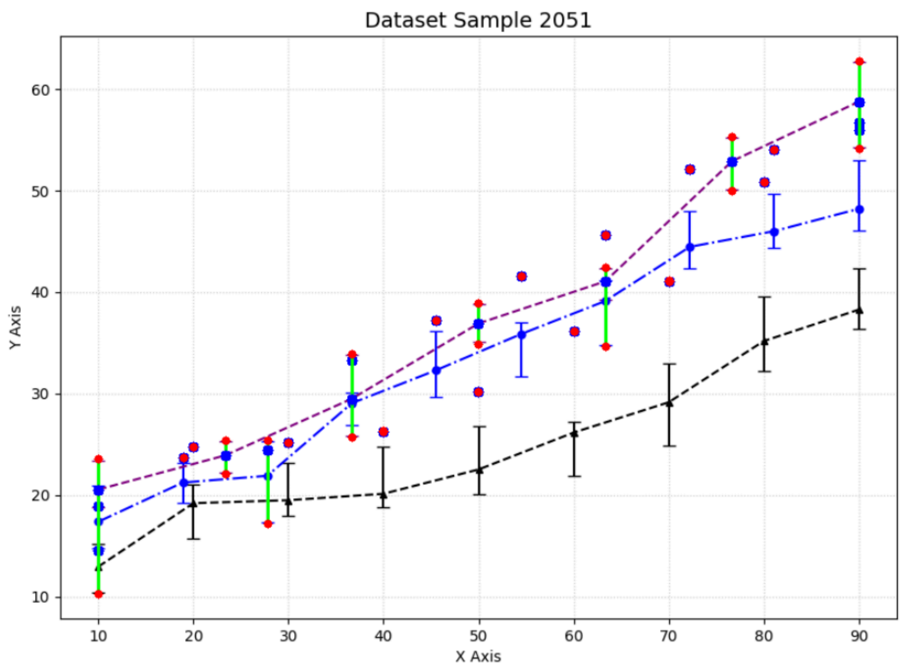
05 Quantitative and Qualitative Evaluation

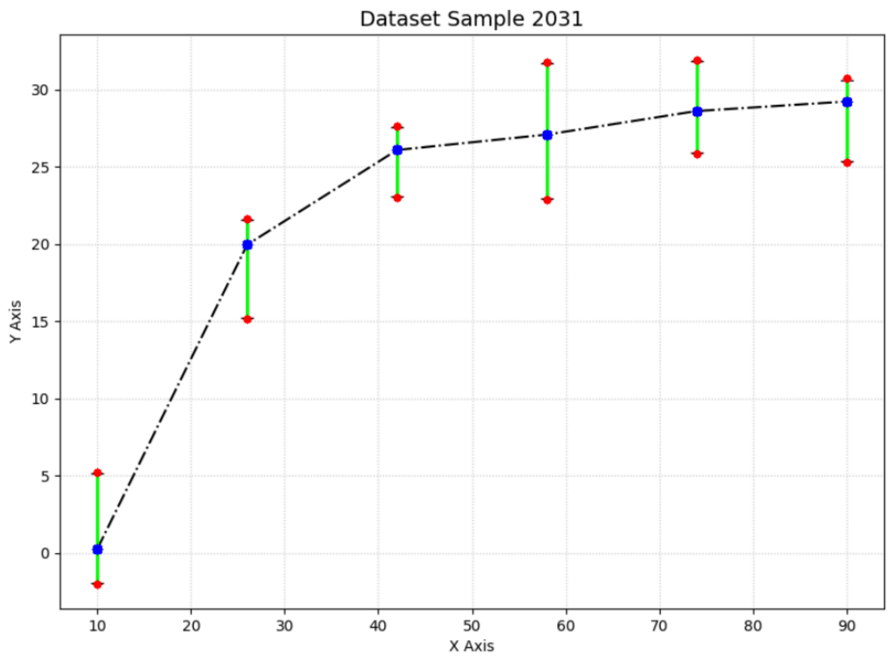
The pipeline underwent a comprehensive evaluation involving 112,000 individual data points across the 3,000 images. Quantitatively, the system reported a Mean Absolute Error (MAE) of

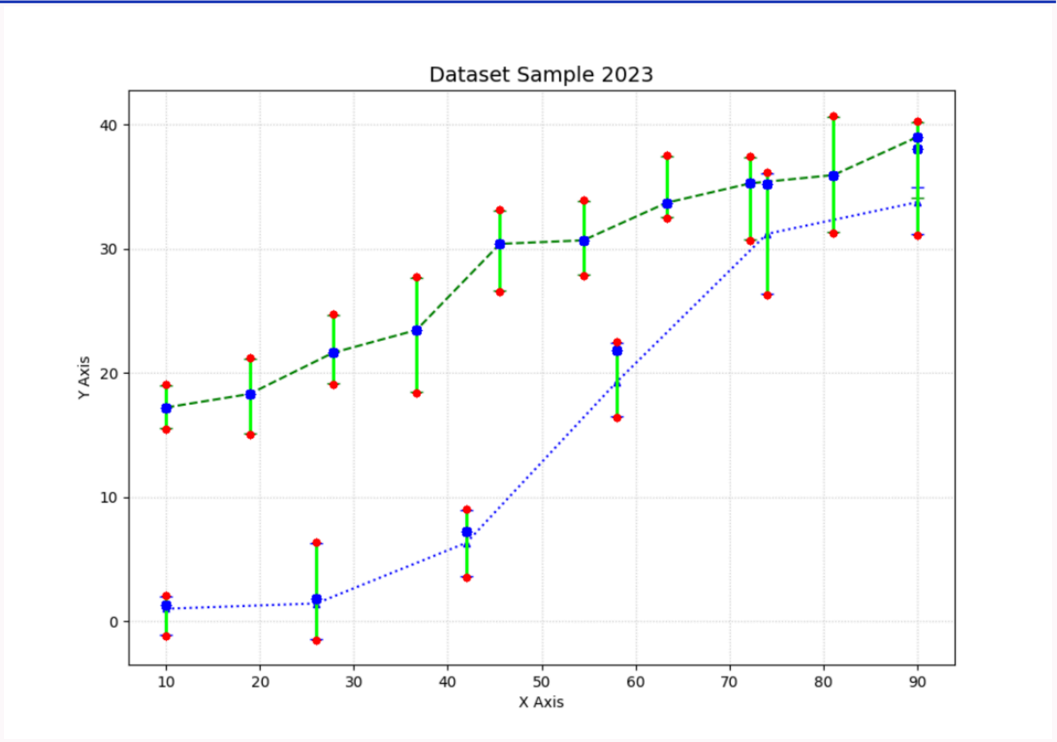
21.98 pixels and an accuracy of **35.63%** within a strict **±2.0-pixel** tolerance. While these figures highlight the precision challenges inherent in automated plot parsing, they provide a baseline for understanding algorithmic limitations in the presence of complex backgrounds. This quantitative rigor follows standards seen in the Journal of Electronic Imaging, where MAE is the preferred metric for spatial accuracy. Qualitatively, a visual verification suite was developed to overlay the detection results back onto the original images. These visualizations, which color-code data points in blue, endpoints in red, and the detected extent in green, confirm that the algorithm successfully identifies error bar structures but is occasionally misled by overlapping graphical elements. This dual-layered evaluation ensures that the system's performance is documented both in terms of raw statistical accuracy and its practical utility in a visual data extraction context.

Some sample visualization of output data:




Dataset Sample 2051
Y Axis
X Axis


Dataset Sample 2031
Y Axis
X Axis



06

LIMITATIONS AND FUTURE WORK

The current code suffers from a number of key shortcomings that will guide the future development of the code. The large MAE is largely due to the interference caused by the horizontal grid lines and axis ticks, which the current vertical profiling algorithm may mistakenly identify as error bar tips. Moreover, the algorithm models error bars as perfectly vertical and data points as centered on them; however, as stated in Nature Methods, in real-world visualizations, offset error bars are sometimes used to avoid visual clutter in busy plots. Future versions of this code will seek to integrate Hough Line Transform or Connected Component Analysis (CCA) algorithms to improve the ability of the algorithm to distinguish between actual error bar tips and other plot features, such as grid lines. The addition of a machine learning "sanity check" may also help the algorithm to disregard non-statistical lines, thereby greatly reducing the MAE.