Line Graph Interpretation Tool for Visually Impaired Users

Ajitesh Parihar, Manpreet Kour, Mohamed Saber

Line Graph Interpretation Tool for Visually Impaired Users*

Ajitesh Parihar
LACL, Université Paris-Est
Créteil, France
0009-0001-3162-1470

Manpreet Kour
LACL, Université Paris-Est
Créteil, France
0009-0007-4356-936X

Mohamed Saber
5G Network Engineer, Orange
Orleans, France
0009-0003-0120-9391

***Abstract*—Line diagrams and mathematical function plots are commonly used in scientific and educational textbooks and articles to convey quantitative relationships. However, their visual nature presents significant challenges and barriers for visually impaired learners. Despite the ongoing efforts to improve accessibility in higher education, the non-text content still remains difficult to interpret non-visually. The existing diagram analysis approaches often focus on chart type classification, require manual user intervention for tasks such as label mapping, or fail to provide accessible end-to-end interaction for non visual analysis. Also, many of these systems do not explicitly consider the requirements of visually impaired users, resulting in workflows and interfaces that are not accessible. In this paper, we propose an accessibility aware framework for automated analysis of line diagrams from raster images. The proposed end-to-end system performs CNN based curve segmentation, diagram type classification, label and tick extraction using optical character recognition, mapping of curve pixels to the diagram domain, and supports interactive querying along with a user interface compliant with the Web Content Accessibility Guidelines. Evaluation is conducted using a large synthetic dataset of 2D chart images representing a variety of mathematical functions and discrete point line charts, which is used for both model training and system-level assessment.**

***Index Terms*—chart parsing, line chart data extraction, computer vision, optical character recognition, accessibility, assistive technology.**

I. Introduction

Line diagrams and mathematical function plots are widely used to communicate quantitative relationships in scientific, engineering, and educational materials. These visual representations play an essential role in conveying trends, functional behavior, and comparative information. However, their reliance on visual perception presents significant accessibility challenges for visually impaired learners. This is because key information such as axes, labels, and curve relationships is embedded within images rather than accessible data structures, and the positioning of this information can vary significantly across diagrams.

Despite ongoing efforts to improve accessibility in higher education, visual materials found in textbooks, lecture slides, and online resources remain difficult to interpret non-visually. While alternative text and static descriptions can provide limited context, they often require deliberate manual effort and are insufficient for enabling detailed exploration and interaction with graphical content. As a result, visually impaired students frequently lack independent access to essential graphical information required for learning mathematics, science, and engineering disciplines.

Existing research on chart understanding and diagram analysis has primarily focused on general chart classification, visual element detection, or data extraction accuracy. Many approaches require manual user intervention for tasks such as axis calibration, label mapping, or data point annotation, limiting scalability and usability. Other fully automated systems prioritize visual parsing or reasoning performance but do not explicitly address accessibility requirements or provide user interfaces designed for non-visual interaction. Consequently, these systems fail to support accessible, end-to-end analysis of line diagrams.

In our previous work [1], we presented an overview of a diagram classification pipeline aimed at improving accessibility for visually impaired users and introduced tools for generating datasets to support machine learning-based diagram understanding. That work focused on high-level diagram categorization and the initial steps required for developing scalable diagram analysis tools.

In this paper, we extend our prior research by focusing specifically on the automated analysis of line diagrams and mathematical function plots. We present an accessibility-aware, end-to-end framework that integrates visual parsing, data reconstruction, and accessible interaction. The proposed system is designed to enable visually impaired users to independently analyze line plots by extracting underlying numerical relationships and supporting non-visual exploration through an accessibility-compliant interface.

II. Existing Works

Within our broader research project, we presented [2] a framework for making UML class diagrams accessible to visually impaired users. The system combines YOLOv8-based object detection, dual optical character recognition pipelines, and shallow formal semantics to automatically reconstruct diagram elements, including classes, attributes, methods, and relationships. Extracted information is represented in structured formats and exposed through a query-based interface that supports accessible text and audio output.

Automatic extraction and understanding of chart data from raster images has received significant attention due to the

widespread use of visualizations in scientific, educational, and technical documents. Early research primarily adopted hybrid pipelines that combine deep learning-based visual detection with rule-based reasoning to recover structured data from charts. ChartOCR exemplifies this approach by employing convolutional neural networks to detect chart components such as axes, legends, and data marks, followed by heuristic post-processing to reconstruct numerical values across multiple chart types [3]. While effective, such methods rely heavily on handcrafted rules, which can limit robustness and adaptability to unseen layouts or noisy inputs.

More specialized solutions have been proposed for extracting data from scientific plots, particularly line charts. LINEEX introduces a modular framework designed for scientific line graphs, integrating keypoint detection, curve extraction, and grouping strategies to handle dense and overlapping curves commonly found in research publications [4]. Although this targeted design improves performance for specific chart types, the pipeline nature of the approach makes it susceptible to error propagation across stages and requires careful tuning for different visualization styles.

Recent work has explored end-to-end learning paradigms that avoid explicit chart component modeling. DePlot reframes chart understanding as a plot-to-table translation problem, directly converting chart images into linearized tabular representations that can be processed by large language models for downstream reasoning [5]. This approach reduces reliance on handcrafted heuristics and demonstrates strong performance in few-shot and reasoning-based tasks. However, DePlot prioritizes semantic interpretation over precise numerical extraction, limiting its suitability for applications requiring high-fidelity data recovery.

Beyond individual chart extraction, broader systems have addressed the problem of parsing figures in scientific documents. FigureSeer represents an early end-to-end framework for detecting, classifying, and extracting data from result figures in research papers [6]. By combining convolutional feature learning with structural analysis, FigureSeer enables large-scale indexing of quantitative findings. Nevertheless, its document-centric focus does not explicitly address user interaction or accessibility considerations.

Accessibility-oriented research has emerged to address the limitations of purely visual chart representations. Chart4Blind specifically targets visually impaired users by converting chart images into accessible formats such as SVG, CSV, and textual descriptions that are compatible with screen readers and tactile devices [7]. While this work makes an important contribution toward inclusive visualization access, it primarily focuses on format conversion and depends on the reliability of underlying chart parsing methods, without offering an integrated environment for interactive data analysis.

General-purpose optical character recognition systems play a critical role in chart understanding by extracting textual elements including axis labels, tick values, and legends. PaddleOCR provides a state-of-the-art OCR framework with an integrated chart parsing module that combines visual and textual cues using vision–language modeling techniques [8]. Although such systems improve robustness across diverse layouts, their performance remains sensitive to chart quality and resolution, and they do not explicitly address accessibility or user interaction beyond data extraction.

In parallel with fully automated approaches, interactive and human-assisted tools remain widely used in practice. WebPlotDigitizer enables users to manually calibrate axes and extract numerical data points from chart images, offering high accuracy at the cost of significant user effort [9]. Similarly, commercial services such as the Graph2Table API provide automated graph-to-table conversion workflows designed for practical integration, though their internal processing pipelines are not transparent [10].

More recently, large language models such as ChatGPT have been explored as multimodal interfaces for interacting with chart data. When combined with visual inputs or structured representations, these models can answer questions, summarize trends, and assist exploratory analysis [11]. However, current LLM-based approaches are not designed for precise chart digitization and may hallucinate or misinterpret numerical values, limiting their reliability for scientific data extraction.

Despite these advances, several existing solutions are not fully automatic and require user intervention, such as manually mapping axes, labels, or data points, which limits scalability and usability. Conversely, fully automated approaches primarily focus on data extraction or reasoning accuracy and often overlook the design of accessible user interfaces that support end-to-end analysis of chart images, particularly for visually impaired users. As a result, current systems rarely integrate robust chart understanding with accessibility-aware interaction and exploration capabilities within a single framework.

III. Line Graphs Domain

There is a wide variety in the type of line graphs we can find throughout the worlds of academics, business, and media. There also exists a difference in the methods used to produce the data points in the graph. They could be either generated by a mathematical function or collected from observation on specific topics.

A. Function Plots

These are the graphs that are plotted based on the results produced by a mathematical function. It is possible for each type of function to produce a visually distinct graph, with a variety in number of lines, curves, intercepts, turning points and inflection points. We took the most common types of functions that we encounter in academics which have distinct shapes to recognize and classify. These include:

Polynomial functions:

- **Linear:** $f(x) = ax + b$ — a straight line where a determines the slope and b determines the vertical intercept.
- **Quadratic:** $f(x) = ax^2 + bx + c$ — a parabola where a controls the opening direction and width, while b and c determine the horizontal position and vertical offset.

- **Cubic:** $f(x) = ax^3 + bx^2 + cx + d$ — a third-degree polynomial capable of exhibiting turning points and an inflection point, producing an S-shaped curve depending on the coefficients.
- **Quartic:** $f(x) = ax^4 + bx^3 + cx^2 + dx + e$ — a fourth-degree polynomial that may contain multiple turning points and changes in concavity depending on the coefficients.
- **Quintic:** $f(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$ — a fifth-degree polynomial with potentially complex curvature and multiple inflection points.

Absolute and radical functions:

- **Absolute value:** $f(x) = a|x+h|+k$ — a V-shaped graph where a controls the steepness of the arms, h shifts the vertex horizontally, and k shifts it vertically.
- **Square root:** $f(x) = a\sqrt{x+b}$ — a radical function whose starting point is shifted horizontally by b, while a controls the direction and vertical scaling of the curve.

Exponential and logarithmic functions:

- **Exponential:** $f(x) = ae^{\,b(x+h)} + k$ — an exponential function where a controls vertical scaling, b controls growth or decay rate, h shifts the curve horizontally, and k shifts it vertically.
- **Logarithmic:** $f(x) = a\log(x+b)$ — a logarithmic function where a controls vertical scaling and b shifts the domain horizontally.
- **Reciprocal:** $f(x) = \frac{a}{x+h} + k$ — a rational function with vertical asymptote at $x = -h$ and horizontal asymptote at $y = k$, while a controls vertical scaling.

Trigonometric and hyperbolic functions:

- **Sinusoidal:** $f(x) = a\sin(bx+c)+d$ — a periodic wave where a determines amplitude, b determines frequency, c controls phase shift, and d shifts the midline vertically.
- **Tangent-like:** $f(x) = a\tan(bx+c)+d$ — a periodic function with vertical asymptotes, where a controls scaling, b controls frequency, c controls phase shift, and d shifts the graph vertically.
- **Secant-like:** $f(x) = a\sec(bx+c)+d$ — a periodic function consisting of repeating curved segments separated by vertical asymptotes, with parameters controlling amplitude, frequency, phase shift, and vertical offset.
- **Hyperbolic sine:** $f(x) = a\sinh(bx)$ — an odd function with exponential growth in both directions, where a controls scaling and b controls horizontal stretching.
- **Hyperbolic cosine:** $f(x) = a\cosh(bx)$ — an even function forming a symmetric U-shaped curve with a minimum at the center.
- **Hyperbolic tangent:** $f(x) = a\tanh(bx)$ — an S-shaped curve approaching horizontal asymptotes as x approaches positive or negative infinity.

B. Data Plots

These graphs are plotted based on a series of discrete data points which come from measured or collected data. They can be useful to gain insights on the trends, patterns and relationships in the data.

C. Features Specification

Despite having different appearances, all these types of graphs contain specific features that we need to extract, which would allow us to execute further queries. These include:

- Graph class: linear, quadratic, exponential...
- X axis range (min and max values)
- X axis type (numeric / time series)
- X axis scale (linear / logarithmic)
- Y axis range (min and max values)
- Y axis scale (linear / logarithmic)
- List of extracted points coordinates

After we have the class of graph and list of extracted points mapped to the domain represented by the axes, we can calculate line graph type specific features. These would include:

1) Linear Function Graphs: For linear functions, the extracted points are used to estimate the line equation in slope-intercept form $y = mx + b$. The following features are supported:

- Slope (m) – measures the rate of change between x and y.
- Trend – describes whether the line is increasing, decreasing, or constant.
- Y-intercept (b) – the point where the line crosses the y-axis.
- Equation – reconstructed linear equation estimated from sampled points.

2) Quadratic Function Graphs: For quadratic graphs, a second-degree polynomial of the form $y = ax^2 + bx + c$ is fitted. The extracted coefficients and derived parameters describe key geometric and algebraic characteristics:

- Coefficients (a, b, c) – parameters of the polynomial.
- Vertex – the turning point of the parabola, representing the minimum or maximum value.
- Concavity – whether the parabola opens upward or downward.
- Axis of symmetry – vertical line passing through the vertex ($x = -\frac{b}{2a}$).
- Roots or X-intercepts – solutions to $ax^2 + bx + c = 0$.
- Y-intercept – where the curve crosses the y-axis ($x = 0$).

3) Cubic and Higher-Order Polynomial Graphs: For cubic, quartic, and quintic functions, higher-degree polynomial fitting is used to extract advanced shape descriptors:

- Polynomial coefficients – estimated parameters of the fitted function.
- Turning points – local maxima and minima found from first-derivative analysis.
- Inflection points – points where curvature changes sign.
- Concavity – describes the curvature and direction of bending.
- Roots – real zeros of the polynomial function.
- Y-intercept – value of the function at $x = 0$.
- End behavior – describes how the function behaves as $x \to \pm\infty$.

4) Absolute Value and Square Root Graphs: For piecewise and root-based graphs, distinctive structural features are computed:

- Vertex – the corner or starting point of the graph.
- Slopes – left and right gradients for absolute value functions.
- Direction – whether the graph opens upward or downward.
- Domain and range – estimated intervals of valid input and output values.
- Intercepts – intersection points with the coordinate axes.
- Estimated equation – reconstructed analytical form based on sampled data.

5) Exponential and Logarithmic Graphs: For exponential and logarithmic functions, exponential regression or logarithmic fitting is applied:

- Estimated base and coefficient – parameters controlling growth rate and amplitude.
- Growth or decay behavior – whether the function increases or decreases.
- Asymptotes – horizontal or vertical lines the graph approaches but never reaches.
- Domain and range – valid input and output intervals.
- Intercepts – points of intersection with the axes.
- Estimated equation – fitted model representation.

6) Reciprocal Function Graphs: Reciprocal functions exhibit asymptotic and symmetric behavior:

- Estimated coefficient – scalar factor in $y = \frac{k}{x}$.
- Horizontal and vertical asymptotes – boundaries the function approaches.
- Curvature and quadrant placement – identifies the branches of the hyperbola.
- Domain and range – intervals excluding undefined points.
- Intercepts and equation estimation.

7) Trigonometric Function Graphs: For periodic functions such as sine, cosine, tangent, and secant-like curves, the following periodic parameters are computed from the extracted points:

- Amplitude – maximum displacement from the midline.
- Midline – central axis about which oscillation occurs.
- Period and frequency – cycle length and number of repetitions per unit.
- Phase shift – horizontal translation relative to the origin.
- Asymptotes (for tangent and secant) – locations of discontinuities.
- Intercepts and estimated equation.

8) Discrete and Multi-Line Graphs: For graphs composed of discrete points or multiple line segments:

- Recovered data points – set of all detected coordinates.
- Number of discrete points or lines – structural count features.
- Overall trend and volatility – describes consistency or variation.
- Maximum and minimum y-values – identifies peaks and troughs.

9) Scatter Plots: Scatter plots are analyzed statistically to describe data distribution:

- Pattern type – overall distribution form (linear, clustered, random).
- Correlation – strength and direction of relationship between x and y.
- Clustering behavior – whether data points form distinguishable groups.
- Cluster count and quality – derived using silhouette scores.

IV. Dataset and Hardware Specification

A key requirement for training and evaluating models for line diagram and function plot analysis is the availability of a dataset that captures a wide variety of diagram classes and visual styles. While chart images are readily available online, such images are often unclassified, inconsistently styled, and not available at resolutions suitable for machine learning training or reliable optical character recognition (OCR). In addition, publicly available datasets frequently lack structured annotations for critical elements such as axes, labels, and curves. Existing datasets, such as the Function_Graphs dataset available on Kaggle [12], provide limited coverage of function types and visual variations, and often suffer from missing axis or curve annotations or insufficient image resolution for downstream tasks. These limitations motivated the development of a tool for the automated generation of synthetic graph images, enabling controlled variation in visual styles, function types, and annotation quality to support both model training and system-level evaluation.

A. Dataset Generator

To generate a diverse and well-controlled dataset of line diagrams, we developed a fully automated dataset generation pipeline for both mathematical function plots and data-driven line charts. For mathematical function plots, the system uses the SymPy symbolic mathematics library [13] to define and manipulate functions from predefined families, including linear, polynomial, radical, absolute value, and trigonometric functions. Each function is represented symbolically and parameterized using randomly sampled coefficients to introduce variability across generated instances.

Even within a single function family, different coefficient values can result in a wide range of visual appearances, and in some cases produce plots that resemble other function types. To ensure that generated plots remain representative of their intended function classes and resemble diagrams commonly found in educational materials, we restrict the domains of both coefficients and independent variables. A safe plotting range is inferred automatically by analyzing the symbolic expression to avoid undefined regions, discontinuities, or invalid values. The function is then sampled numerically over this domain to obtain dense point representations suitable for visualization.

For data-driven line plots, randomization is used to model the relationship between the independent and dependent variables, allowing generation of discrete point-based line charts

that differ from analytical function plots while preserving realistic structural properties. Using the sampled data, standardized plot images are rendered with controlled visual styles, including grid configuration, axis placement, and labeling.

In addition to rendering the final images, the pipeline generates pixel-level annotation masks that separately capture the function curve, coordinate axes, and grid lines. For each generated diagram, structured metadata—including the function type, symbolic expression, coefficient values, and plotting range—is stored in JSON format. This automated process enables scalable generation of high-quality, fully labeled datasets without manual annotation, making it well suited for training and evaluating machine learning models for line diagram understanding and accessibility oriented applications. The final dataset contains 15,000 graph images spanning 15 distinct classes, with 1,000 samples per class to ensure a perfectly balanced distribution.

B. Hardware Resources

We used HPC (Hight Performance Computing) resourcesresources provided by CRIANN (Centre Régional Informatique et d'Applications Numériques de Normandie) through project number 2024004. Experiments were conducted on the Austral HPDA cluster, utilizing a GPU configuration with NVIDIA A100 hardware via Multi-Instance GPU (MIG) partitioning. Specifically, a MIG instance with 4 CPU cores, 12 GB of system memory, and a single NVIDIA A100 GPU partition with 20 GB of GPU memory, 28 streaming multiprocessors (SMs), and 108 tensor cores was allocated for model training and evaluation. These resources enabled efficient handling of large-scale datasets and accelerated deep learning workloads.

V. Feature Extraction Pipeline

The proposed system employs a multi-stage feature extraction pipeline designed to robustly analyze the images with varying visual styles. Rather than relying on rule-based extraction, which is not versatile, the pipeline integrates machine learning models and structured post-processing to extract visual, textual, and semantic information required for accessible analysis.

A. Curve Segmentation Model

Accurate separation of the plotted curve from other visual elements is a fundamental requirement for reliable line diagram analysis. Classical computer vision approaches based on edge detection or thresholding are highly sensitive to variations in visual style, line thickness, grid density, and rendering quality, making them unreliable across diverse diagrams. To address these limitations, curve extraction is formulated as a semantic segmentation problem and solved using a learning-based approach.

Semantic segmentation of graph components is performed using a UNetResNet50 architecture implemented in PyTorch. The model combines a ResNet-50 encoder, initialized with ImageNet-pretrained weights, and a U-Net–style decoder with skip connections to preserve fine-grained spatial detail. The network performs multi-class pixel-wise classification to generate separate masks for the function curve, coordinate axes, grid lines, and background.

During training, the encoder weights are frozen to leverage pretrained visual features while reducing overfitting and training cost. Input images are resized to a resolution of 256×256 pixels and normalized using ImageNet mean and standard deviation values. The model is optimized using the Adam optimizer with a learning rate of 1×10^{-4} and trained for five epochs. Model performance is monitored using validation loss, Dice coefficient, and Intersection over Union (IoU), and the best-performing checkpoint is selected based on validation loss. Compared to classical computer vision techniques, this learning-based segmentation approach provides more reliable and robust curve extraction across varying visual styles and diagram configurations1.

B. Optical Character Recognition for Labels and Ticks

Textual elements such as axis tick labels and numerical annotations provide essential semantic context for interpreting line diagrams. To extract this information, we employ PaddleOCR, a state-of-the-art optical character recognition framework designed for robust text detection and recognition in natural and structured images. PaddleOCR is applied to regions in close proximity to the detected coordinate axes, as identified by the segmentation masks, to focus recognition on relevant textual content.

Prior to OCR, image preprocessing steps such as grayscale conversion, contrast normalization, and noise reduction are applied to improve recognition accuracy, particularly for small or densely packed tick labels. The recognized text is parsed to identify numerical values and is associated with corresponding spatial locations along the axes. This association enables the establishment of semantic anchors that relate pixel positions to numerical values.

Together, the segmentation masks produced by the UNetResNet50 model and the textual information obtained via PaddleOCR enable reliable localization of the coordinate axes and inference of axis scales. This combined visual–textual understanding supports accurate pixel-to-value mapping and is a key component in reconstructing the underlying numerical representation of the plotted function from the graph image.

C. Diagram Classification Model

Function classification is performed using a Vision Transformer Tiny (ViT-Tiny) model trained to predict the underlying function class of a line diagram from the extracted curve representation. Using curve segmentation masks as input reduces sensitivity to visual style variations such as grid density, color schemes, and background artifacts, allowing the classifier to focus on the geometric structure and shape of the curve, which is more indicative of function type.

The ViT-Tiny architecture processes the input as a sequence of fixed-size image patches and learns global contextual relationships across the diagram. This global receptive field is well suited for capturing overall curve behavior and functional

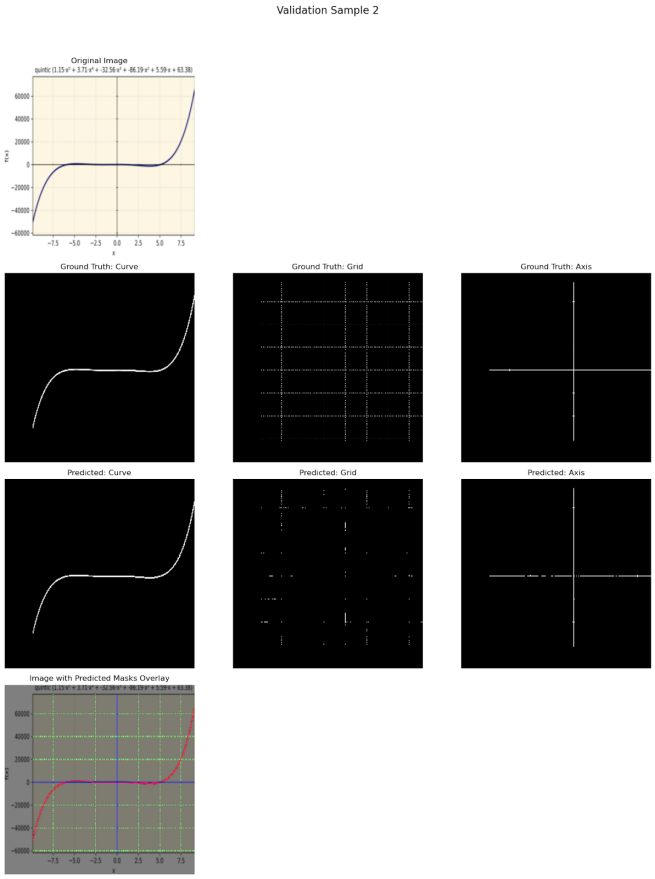

Fig. 1. Example outputs of the curve segmentation model displaying the ground-truth segmentation masks for the function curve, coordinate axes, and grid lines; predicted segmentation masks produced by the UNetResNet50 model; and the original input image with predicted masks overlaid. The results illustrate accurate separation of graph components across overlapping visual structures.

trends. All input masks are resized to a resolution of 224×224 pixels and normalized using ImageNet mean and standard deviation values to ensure compatibility with pretrained transformer weights.

The classifier is initialized from a pretrained ViT-Tiny backbone, and the final classification head is replaced to match the number of function classes in the dataset. Training is conducted in a supervised multi-class setting using cross-entropy loss. The dataset is randomly split into training and validation subsets, and optimization is performed using the Adam optimizer with a learning rate of 1×10^{-4}. Model performance is monitored using validation loss and validation accuracy, and the best-performing checkpoint is selected based on validation accuracy.

The classification model is trained independently of the segmentation network and can operate prior to or alongside detailed curve reconstruction. Within the overall pipeline, the ViT-Tiny classifier provides high-level semantic identification of the function type, while the UNetResNet50 segmentation model supplies pixel-level structural information. Together, these complementary components enable both coarse-grained identification and fine-grained reconstruction of line diagrams, supporting robust and accessible interpretation of graphical content.

D. Mapping Curve to Diagram Domain

Mapping visual information to numerical meaning is a critical step in transforming line diagrams into accessible and interpretable representations. To bridge the gap between image-space representations (pixels, masks, and bounding regions) and data-space representations (numerical coordinates and curve points), we employ a dedicated curve-to-domain mapping pipeline. This process enables reconstruction of the underlying mathematical relationship solely from the segmented diagram image.

The mapping pipeline assumes as input the original plot image, pixel-level segmentation masks for the curve, coordinate axes, and grid lines, as well as metadata generated during dataset creation, including image resolution and the numerical domain and range of the plot. All images and masks are loaded and converted into normalized numerical arrays, establishing a consistent image coordinate system in which pixel locations can be analyzed and transformed.

A key step in the pipeline is the detection and localization of the coordinate axes. The segmented x-axis and y-axis masks are analyzed independently to identify dominant horizontal and vertical line structures, respectively. The intersection of these detected axes is computed to determine the origin of the coordinate system in pixel space. This anchor point is essential, as pixel positions have no numerical interpretation without a known reference.

Once the axes are localized, the mapping between pixel distances and numerical values is inferred. Using known axis limits provided by metadata and measured pixel distances between grid lines or axis tick marks, linear scale factors are computed for both dimensions. These scale factors define the number of pixels per unit along the x and y axes and enable a reversible transformation between pixel coordinates and numerical values.

The curve segmentation mask is then processed to extract all foreground pixels corresponding to the plotted curve. These pixels are filtered to remove noise, sorted along the horizontal direction, and optionally thinned to produce a clean, single-pixel-wide trace. Each curve pixel is subsequently mapped into numerical coordinates by computing its offset from the detected origin and applying the inferred scale factors. This results in a set of sampled data points that approximate the original mathematical function or data-driven relationship represented in the diagram.

To ensure robustness across diverse function types, additional logic is applied to handle discontinuities and outliers. The pipeline detects abrupt vertical jumps, removes extreme values, and splits curves into continuous segments when necessary. This prevents incorrect connections across asymptotes and preserves the structural properties of functions such as trigonometric or rational expressions. Through this process,

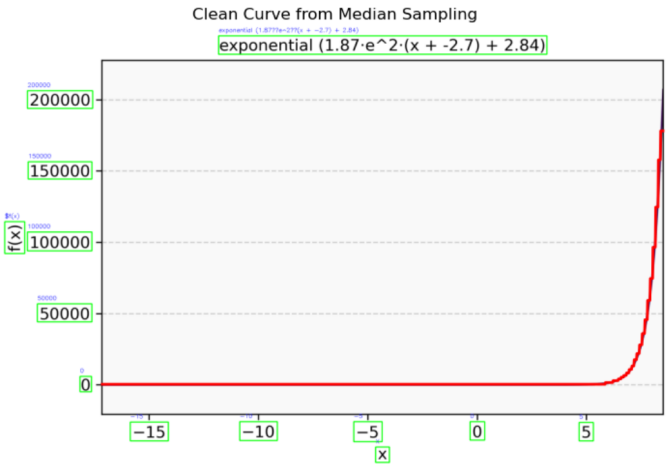

Fig. 2. Visualization of text extraction and curve-to-domain mapping. Green bounding boxes indicate axis tick labels and textual elements detected using PaddleOCR. The red polyline represents reconstructed curve points obtained by mapping segmented curve pixels from image space to the numerical diagram domain. This visualization demonstrates the integration of segmentation, OCR, and coordinate mapping for reconstructing the underlying function from the image.

the system reconstructs a faithful numerical representation of the plotted curve, enabling downstream analysis, querying, and accessible interaction2.

To make the curve-to-domain mapping explicit, the extracted textual, structural, and numerical information is packaged into a unified data structure. This representation combines OCR-derived labels, axis tick information, and reconstructed curve points, enabling downstream querying and accessible interaction. Listing 1 illustrates how this information is stored.

Listing 1. Structured extracted graph information

```
{
    "title": {
        "text": title_text,
        "box": title_box
    },
    "x_label": {
        "text": x_label_text,
        "box": x_label_box
    },
    "y_label": {
        "text": y_label_text,
        "box": y_label_box
    },
    "x_ticks": [
        {"text": txt, "box": box} for txt,
            box in x_ticks
    ],
    "y_ticks": [
        {"text": txt, "box": box} for txt,
            box in y_ticks
    ],
    "curve_points": [
        {"x": float(x), "y": float(y)}
        for x, y in zip(x_axis_vals,
                        y_axis_vals)
    ]
}
```

E. Query System and Response Generation

To enable accessible and flexible interaction with extracted diagram content, the system provides a keyword-driven query interface that allows users to request both high-level summaries and detailed analytical information. The query mechanism is designed to support non-visual exploration by mapping natural-language user input to structured features derived from the diagram analysis pipeline.

Queries are organized into two categories: global queries and graph-type-specific queries. Global queries apply across all diagram types and include requests related to graph identification, confidence estimation, summary descriptions, and available query options. In contrast, graph-type-specific queries are dynamically selected based on the predicted diagram class and enable access to properties relevant to that function family. For example, linear graphs support queries related to slope and intercepts, while quadratic and higher-order polynomials support queries involving vertices, roots, concavity, and symmetry. Trigonometric, exponential, logarithmic, and hyperbolic functions expose queries related to periodicity, asymptotes, amplitude, and growth behavior, while data-driven plots such as line and scatter charts support statistical and structural queries.

User input is processed using keyword matching against predefined synonym sets, allowing multiple phrasings to map to the same underlying feature. This design accommodates variations in user vocabulary and reduces the need for rigid command syntax. Once a query is matched to a feature, the corresponding response is generated using structured templates that combine extracted numerical data, symbolic properties, and metadata. Responses are formulated as complete textual descriptions suitable for screen reader output and are designed to be concise, informative, and consistent across query types.

The query system is tightly integrated with the feature extraction pipeline, leveraging outputs from segmentation, classification, OCR, and curve-to-domain mapping. This integration enables responses to be grounded in both visual structure and reconstructed numerical representations. By dynamically adapting available queries based on diagram type and extracted features, the system provides an extensible and user-centered mechanism for accessible analysis of line diagrams and mathematical function plots.

VI. Accessible User Interface

To provide an accessible UI (user interface) which is compatible with screen readers and other accessibility tools, we took the web-app approach. It uses React [14] for the front-end and Flask [15] for the backend. Flask was chosen to be able to easily integrate the code from our machine learning training notebooks.

The UI is designed to be easily navigable without using a mouse, as most visually impaired computer users use their

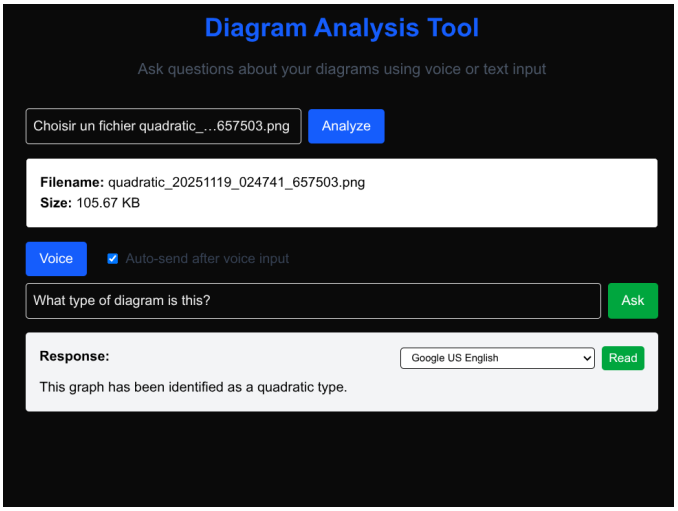

Fig. 3. User-Interface screenshot showcasing the user asking a query using voice input.

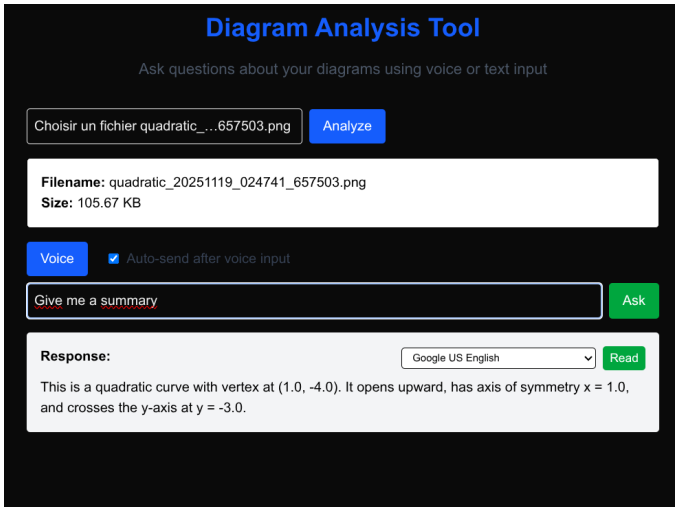

Fig. 4. User-Interface screenshot of the generated output of a user query summarizing the analyzed details.

keyboard or other devices for navigation by going over each item present on the current page. The UI only contains necessary elements, and they are arranged sequentially based on their role in the workflow.

The user follows these steps to analyze a diagram:

1) Image Upload Field: The interface provides a field to upload the diagram image. When the user selects this field, the system file picker is launched to select and upload the image.
2) Analyze Button: When this button is pressed, the selected image is uploaded to the backend and goes through the analysis pipeline. The results of the analysis are stored in the backend as part of the user's session.
3) Query Text Field And Voice Input: After the image is analyzed, the user will be presented with a list of valid queries that can be asked for that specific graph type. The user can use the text field to type their query or press the voice button to use their microphone. Figure 1 demonstrated an example of the user asking a query about the class of diagram.
4) Output: This field provides the answer of the query to the user. It is also used to present the valid queries after the initial analysis of an image. Whenever this field is updated, the screen reader is focused on this field to read out the output. We have also provided a button which can be used to read out the output using the browser's TTS (Text-to-Speech), for users who do not have a screen reader installed 4.

The voice input and output speech synthesis use the Web-SpeechAPI [16] from the browser. This allows for compatibility across devices and browsers.

VII. Results and Evaluation

This section evaluates the proposed framework at both the component level and the system level. The evaluation focuses on the accuracy and robustness of curve segmentation, function classification, and curve-to-domain mapping, as well as the effectiveness of the integrated pipeline in supporting accessible, end-to-end analysis of line diagrams and mathematical function plots.

The performance of the semantic segmentation model is assessed on a held-out validation subset of the synthetic dataset. Evaluation emphasizes accurate separation of the function curve, coordinate axes, and grid lines, which is essential for downstream processing stages. Standard overlap-based metrics, including Dice coefficient and Intersection over Union (IoU), are used to quantify segmentation quality. Qualitative inspection further shows that the UNetResNet50 model reliably distinguishes the plotted curve from visually similar structures such as grid lines and axes across a wide range of visual styles. Representative segmentation outputs and overlays are shown in Figure 1, illustrating the robustness of the learning-based approach.

Function classification performance is evaluated on a synthetic test set comprising 3,521 samples spanning sixteen function classes. The Vision Transformer Tiny (ViT-Tiny) classifier was trained on curve segmentation masks, and as show in Figure 5 achieves a test accuracy of 0.9952. Precision, recall, and F1-score are consistently high across all function classes, with macro-averaged and weighted-averaged F1-scores of 1.00. Most function categories, including linear, exponential, logarithmic, polynomial, and radical functions, achieve near-perfect classification performance. Slightly lower per-class accuracy is observed for function types with visually similar shapes or asymptotic behavior, such as quadratic, secant-like, sinusoidal, and hyperbolic functions; however, accuracy

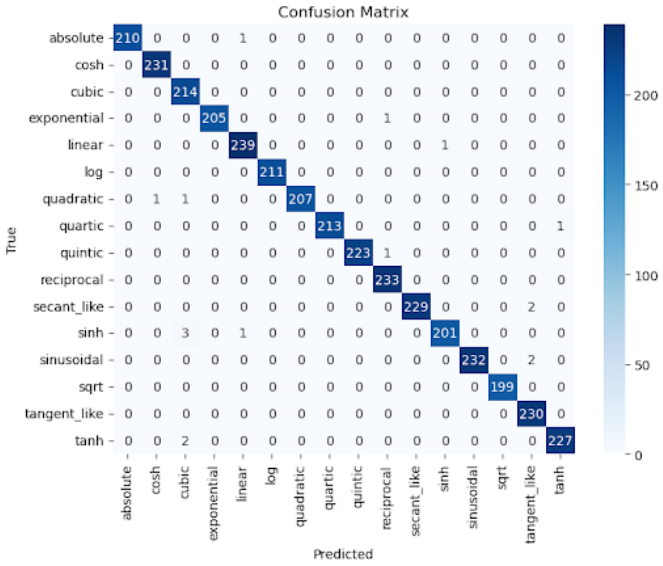

Fig. 5. Confusion matrix of the ViT-Tiny function classifier evaluated on the synthetic test dataset. The matrix illustrates strong diagonal dominance across all sixteen function classes, indicating high classification accuracy. Minor confusion is observed between visually similar function types with overlapping geometric characteristics.

for these classes remains above 0.98. These results indicate that curve-based representations provide sufficient information for reliable semantic classification while remaining robust to stylistic variations.

The accuracy of curve reconstruction and domain mapping is evaluated by examining the numerical data recovered from segmented curves. Using detected axes, OCR-extracted tick labels, and inferred scale factors, curve pixels are mapped from image space to numerical coordinates in the diagram domain. Qualitative evaluation is performed by overlaying reconstructed curve points on the original plot images, as shown in Figure 2. The reconstructed curves exhibit strong alignment with the original plots, preserving both global trends and local structure. Additional handling of discontinuities and outliers improves robustness for functions with non-continuous behavior, such as tangent-like and reciprocal functions.

End-to-end system evaluation demonstrates that the integrated pipeline successfully extracts data the required data from images of line diagrams and the query system allows the users to understand the data in the given plot.

Accessibility evaluation focused on verifying that the proposed system supports effective non-visual interaction across realistic usage scenarios. User interface testing was conducted across multiple device types, including desktop, tablet, and mobile form factors, and across commonly used browsers such as Chrome and Firefox. Screen reader compatibility was evaluated using NVDA [17] and Orca [18], ensuring that interface elements, query responses, and dynamically generated content were correctly exposed through accessible markup and announced in a meaningful order.

In addition, the design and evaluation process benefited from collaboration with Mohamed Saber, a blind software engineer, who provided expert feedback on usability and accessibility requirements. This collaboration helped validate compliance with the Web Content Accessibility Guidelines (WCAG) and informed the design of the query system, including which types of queries are most useful for non-visual exploration of line diagrams. Feedback from this consultation guided refinements to interaction flow, response phrasing, and query coverage, contributing to a more practical and user-centered accessible analysis workflow.The proposed solution that we made is compatible with the refreshable braille display 5th generation used by our engineer ,he was able to easily navigate through the accessibility system and understand about the activity diagram by asking questions and reading answers via display.

VIII. Conclusion and Future Work

This paper presented an accessibility-aware, end-to-end framework for analyzing line diagrams and mathematical function plots from raster images. By integrating semantic segmentation, curve-based function classification, optical character recognition, and curve-to-domain mapping, the proposed system transforms visual diagrams into structured and accessible representations without requiring manual intervention. Experimental results on a large synthetic dataset demonstrate reliable segmentation, near-perfect function classification accuracy, and accurate reconstruction of underlying numerical relationships. Accessibility evaluation further confirms that the system supports non-visual interaction across devices, browsers, and screen readers, enabling independent exploration of graphical content by visually impaired users.

Future work will focus on improving the generalization capability of the classification encoder by incorporating textual and numerical information extracted through OCR, allowing joint reasoning over curve geometry and axis semantics. This multimodal integration is expected to improve robustness when handling visually ambiguous or stylistically varied diagrams. Additional efforts will be directed toward enhancing the user interface based on feedback from further accessibility testing, including refining query workflows, response phrasing, and interaction patterns to better support non-visual exploration. The framework will also be extended to support more complex diagram types, such as multi-curve plots and charts with multiple axes or scales. Finally, conducting formal user studies with a broader group of visually impaired participants will be essential for quantitatively evaluating usability, effectiveness, and real-world impact.

Acknowledgement

This work is part of an internship under the INOVISUP (informatique pour les personnes aveugles et malvoyantes de l'enseignement supérieur) project at LACL, Université Paris-Est Créteil Val-de-Marne (UPEC).

I would like to thank Okanagan College, Careers Hub, Dr. Youry Khemelevsky (Computer Science Chair Okanagan

College), and Gaétan Hains (Professor at UPEC/LACL) for their guidance and making this opportunity possible.

I would also like to thank Dr. Karine Gros (Vice-présidente Politique handicap UPEC) and Jean-Luc Dubois-Rande(Présidente de l'université UPEC) for funding this research.

Ce travail a bénéficié des moyens de calcul du mésocentre CRIANN (Centre Régional Informatique et d'Applications Numériques de Normandie). Part of this work was performed using computing resources of CRIANN (Normandy, France) through project number 2024004.

REFERENCES

[1] A. Parihar, G. Hains, M. Kour, K. Cormier, K. Gros, and M. Saber, "Diagram annotation and classification for visually impaired people in higher education," *Authorea Preprints*, 2025.

[2] M. Kour, G. Hains, and M. Saber, "Diagram analysis for visually impaired readers," Ph.D. dissertation, universite paris est creteil val de marne, 2025.

[3] J. Luo, Z. Li, J. Wang, and C.-Y. Lin, "Chartocr: Data extraction from chart images via a deep hybrid framework," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2021. [Online]. Available: https://www.microsoft.com/en-us/research/publication/chartocr-data-extraction-from-charts-images-via-a-deep-hybrid-framework/

[4] V. P. Shivasankaran, M. Y. Hassan, and M. Singh, "Lineex: Data extraction from scientific line charts," *arXiv preprint*, 2025. [Online]. Available: https://shiva-sankaran.github.io/assets/docs/lineex.pdf

[5] F. Liu, J. M. Eisenschlos, F. Piccinno, S. Krichene *et al.*, "Deplot: One-shot visual language reasoning by plot-to-table translation," in *Findings of ACL*, 2023. [Online]. Available: https://aclanthology.org/2023.findings-acl.660.pdf

[6] N. Siegel, Z. Horvitz, R. Levin, S. Divvala, and A. Farhadi, "Figureseer: Parsing result-figures in research papers," in *European Conference on Computer Vision (ECCV)*, 2016.

[7] O. Moured, M. Baumgarten-Egemole, A. Roitberg, K. Muller, T. Schwarz, and R. Stiefelhagen, "Chart4blind: An intelligent interface for chart accessibility conversion," *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2403.06693

[8] P. Contributors, "Chart parsing module," https://www.paddleocr.ai/main/version3.x/module_usage/chart_parsing.html, 2025.

[9] "Webplotdigitizer: Extract data from charts and plots," https://www.energent.ai/use-cases/en/webplotdigitizer, 2025.

[10] "Ai plot digitizer — graph2table api," https://graph2table.com/, 2025.

[11] OpenAI, "Chatgpt," https://openai.com/chatgpt, 2025.

[12] kopfgeldjaeger, "Function graphs (polynomial) dataset," https://www.kaggle.com/datasets/kopfgeldjaeger/function-graphs-polynomial, 2022, accessed: 2025-12-05. [Online]. Available: https://www.kaggle.com/datasets/kopfgeldjaeger/function-graphs-polynomial

[13] A. Meurer, C. P. Smith, M. Paprocki, and *et al.*, "Sympy: Symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.103

[14] Meta Platforms, Inc., "React: A javascript library for building user interfaces," https://react.dev/, 2024, frontend user interface framework.

[15] Pallets Project, "Flask: A lightweight wsgi web application framework," https://flask.palletsprojects.com/, 2024, python web framework.

[16] W3C, "Web speech api," https://www.w3.org/TR/speech-api/, 2023, w3C Recommendation.

[17] NV Access, "Nvda: Nonvisual desktop access," https://www.nvaccess.org/, 2024, screen reader for Microsoft Windows.

[18] GNOME Project, "Orca screen reader," https://wiki.gnome.org/Projects/Orca, 2024, screen reader for Linux-based systems.