LINEEX: Data Extraction from Scientific Line Charts

Shivasankaran V P*, Muhammad Yusuf Hassan*, Mayank Singh
IIT Gandhinagar, Gujarat, India
vp.shivasan@iitgn.ac.in, md.hassan@iitgn.ac.in, singh.mayank@iitgn.ac.in

Abstract

In this paper, we introduce LINEEX *that extracts data from scientific line charts. We adapt existing vision transformers and pose detection methods and showcase significant performance gains over existing SOTA baselines. We also propose a new loss function and present its effectiveness against existing loss functions. In addition, we synthetically created the largest line chart dataset comprising 430K images. The code is available at* `https://github.com/Shiva-sankaran/LineEX`.

1. Introduction

The modern world generates large volumes of data every day, but most of it is unusable due to processing, representation, and storage challenges. Scientific papers, too, contain a large proportion of non-textual content, such as charts and images, that do not serve more purpose than visualization [15]. This non-textual content, if successfully processed, can be used in designing high-quality scholarly search engines [21], machine-generated task-specific leaderboards [20], and scholarly assistants for impaired people [22]. With the advent of deep learning architectures and the availability of large volumes of scholarly datasets, we have witnessed a recent surge in efforts to extract data from scientific charts. However, most of these works suffer from prevalent ML-related challenges such as reproducibility and inaccessibility (see Table 1 for more details). To this end, we aim to develop a chart information extraction system that is entirely reproducible, publicly available and produces high-quality output.

In contrast to existing chart extraction systems that extract information from all possible types of charts, the current work focuses on *line charts*. We propose a system, hereafter LINEEX, that leverages powerful transformer architectures [23] for information extraction from line charts. The proposed system is inspired by vision transformers [8] that have recently outperformed all existing state-of-the-art systems. LINEEX comprises three modules (i) keypoint extraction, (ii) chart element detection and text extraction, and (iii) keypoints grouping, legend mapping and data scaling. We robustly evaluate each module and compare it with the state-of-the-art ChartOCR system [15]. In addition to the proposed system, we created the largest synthetic dataset for line chart information extraction. To summarise, the main contributions of our work are:

- Construction of the largest line chart dataset comprising 430K images incorporating diversity in the number of lines, font size, figure size, placement of legend boxes, line color and marker styles, background, and gridlines.
- Adapting vision transformer architectures for line chart information extraction tasks.
- Showcasing limitations of existing loss functions and proposal of a new loss function that addresses these challenges.
- Legend-to-line mapping by matching line patches with legend markers based on image similarity.

2. Scientific Line Charts

Line charts are the most common form of charts found in scientific literature. Line charts are used to visually represent a series of data points or a mathematical function. The main components of a line chart include (i) *axes*, (ii) *lines*, (iii) *a legend box* and (iv) *a chart title*. Optionally, the major components can be further subdivided into subcomponents. For example, the legend box comprises visual markers (popularly known as *legend marker*) and associated labels (popularly known as *legend text*), which are required for mapping each line to its corresponding text-marker pair. Similarly, axes comprise titles and ticks. Figure 1 showcases and describes a simplistic variant of a line chart comprising the above components.

For example, charts comprising a legend box inside the chart area with two or more columns for listing text-marker pairs, log scaled axes, or inclined axes ticks. Secondly, we find chart variants comprising different stylistic features, such as lines of different thicknesses, dashed or continuous lines, or lines with or without markers. Lastly, differ-

*Equal contribution

System Name	Axes		Lines			Legends		Title	Scaling	Legend-line Mapping	Code-base Availability
	Title	Ticks	Keypoints	Thickness	Dashing	Marker	Text				
FigureSeer [19]	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓
ChartOCR [15]	✓	✗	✓	✓	✓	✗	✗	✓	✓	✗	✓
Linear Programming [14]	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✗
LineEX (ours)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison of existing chart data extraction methodologies for different line chart components.

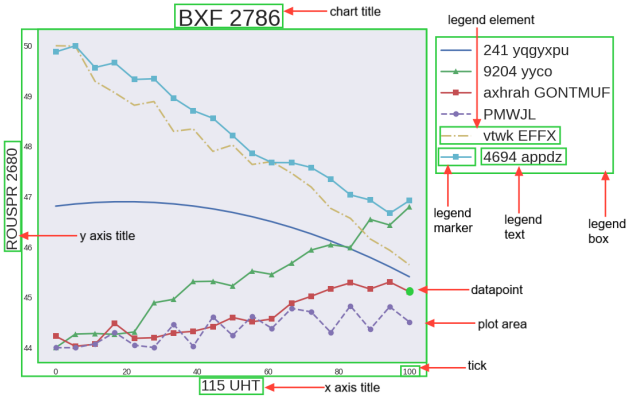


Figure 1: The different components of a line chart.

ent plotting software can generate different visually-looking charts with the same underlying input.

Due to high complexities and diversity, the field of chart data extraction has been relatively less explored. Most of the existing chart datasets (see Table 2 for more details) are synthetic, such as the Adobe Synthetic Chart Dataset [6] and SYN dataset [14]. A few real datasets are curated from the web such as FigureSeer [19], ExcelChart400k [15] and the ICPR-2020 [7]. Real chart datasets are relatively difficult to curate due to problems associated with the manual efforts required to annotate all the components. On the other hand, synthetic chart datasets are generated automatically using plotting software comprising synthetic data values. These datasets are usually more limited in diversity than real datasets but can be constructed in large numbers with minimum human effort.

Given the requirements of extensive manual efforts in curating real datasets, we propose a new line chart dataset. It

Name	**Type**	**Instances**		**Systems**	**Public**
		Full	Line		
FigureSeer [19]	Real	60k	60k	[19][15][14]	only 1k
Adobe Synthetic [6]	Synthetic	198k	4.2k	-	✓
ICPR-2020 [7]	Real	23k	10k	-	✓
ExcelChart400k[15]	Real	400k	122k	[15][14]	✓
SYN[14]	Synthetic	64k	64k	[14]	✗
Our dataset	Synthetic	430k	430k	LineEX	✓

Table 2: Comparison of different chart data extraction datasets.

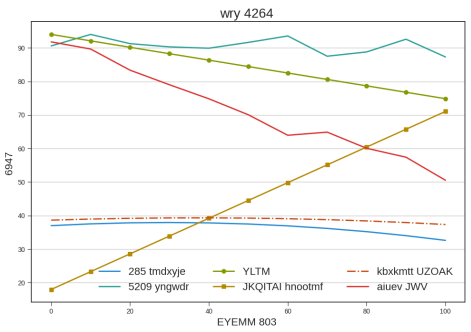
contains 430k line charts generated from Python's popular *Matplotlib library*[1]. Our dataset comprises charts containing 2 to 6 lines. It also contains annotations for bounding boxes for all possible chart components. Diversity in the generation process includes a random number of lines, font, and figure size. We also implement diversity in the placement of legend boxes. Lines have variations in color and marker style (dots, dashes, and shapes). We generate plots with 27 different plotting styles that bring diversity in line markers, backgrounds, and gridlines. The chart text (chart title, axes labels, legend labels) is randomly generated from alphanumeric strings. Further, we segment our dataset into 400,000 images for training, and 10,000 images for validation and 20,000 images for the test.

Table 2 compares our dataset against existing datasets. It is the largest among existing line chart datasets and approximately 4X larger than ExcelChart400k [15]. In contrast to the existing synthetic line chart datasets such as Adobe Synthetic Chart Dataset [6] and SYN dataset [14], our dataset also contains variations in the plotting styles, with charts ranging from darker to brighter backgrounds, bounding boxes around plot area and legend area. Additionally, Adobe Synthetic Chart Dataset does not map the legend marker and text pair to the respective lines. In contrast to the largest existing real dataset ExcelChart400k [15], our dataset also contains annotations for legend markers, legend texts, tick texts and their corresponding boxes. Figure 2 shows two representative line charts from our dataset with different background, tick and legend placement styles.

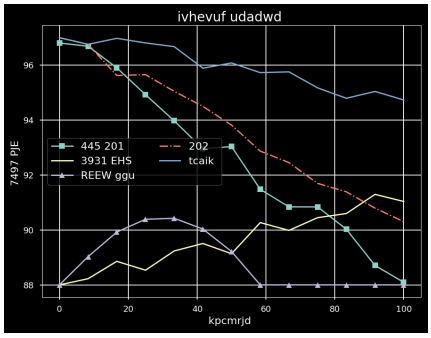
3. Related Work

Most earlier chart data extraction methods used heuristic and rule-based techniques to extract data according to the different chart types. The works by [18] and [10] first classify the chart type using a Support Vector Machine, which is then followed by the use of heuristic based data extraction methods. Such rule-based methods have become less prominent due to the lack of generalisability to real-world data and the need to hand-engineer features. To improve results, some hybrid methods like ChartSense [13] also include some human feedback in the data extraction process. These hybrid methods give better results, but they also use human time and labour and thus cannot be implemented into

[1]https://matplotlib.org



(a) White background with six lines



(b) Dark background with five lines

Figure 2: Representative line charts from our dataset with different plotting styles. The two charts have different legend placement coordinates.

an automated pipeline.

Recently, efforts in deep learning based models have been shown to give better results in chart data extraction. These methods usually build upon object detection models and adapt the processing according to the chart type. ChartText [4] uses a Convolutional Neural Network to classify the chart type, then runs object detection models to extract text and labels. Then, type-specific image processing is done to extract the data. ChartOCR [15] also uses a CNN to classify the chart type, which is then followed by a keypoint detection model to extract certain pre-defined keypoints based on the chart type. Postprocessing is done based on the detected chart type to extract chart data using the identified keypoints. These methods perform better than rule-based methods, but they still give problematic results on real-world data. Also, the ChartOCR model does not allow for legend mapping, i.e. identifying which line corresponds to which legend marker or label. Recently, ChartOCR scores are outperformed by linear programming-based approach [14]. However, the source code and trained models are not publicly available.

4. The LineEX System

The LineEX system comprises a highly modular pipeline. It consists of three main modules: (i) keypoint extraction, (ii) chart element detection and text extraction, and (iii) keypoints grouping, legend mapping and data scaling.

4.1. Keypoint Extraction

This module extracts keypoints that are joined in order to create lines. For keypoint extraction, we adopt the PE-former [17] architecture. PE-former is an end-to-end encoder-decoder transformer architecture proposed for human pose estimation. The best performing PE-former variant comprises Cross-Covariance Image Transformer (XCiT) [3] as the encoder and DETR [5] based transformer decoder. Figure 3 describes the architecture of the keypoint extraction module.

4.1.1 The Architecture

In contrast to the COCO dataset's 17 joint types, LINEEX contains a single class representing a keypoint. Thus, we do not pass the output tokens from the decoder through a feed-forward neural network (FFN) based classification head. However, we leverage FFN to regress each output token to two scalar values (x, y) in the range of [0, 1]. Note that the number of ground truth keypoints in a chart varies drastically based on the number of lines and the complexity of the lines in the chart. A chart with a simple single line typically has 6–10 keypoints, whereas a chart with 4–5 complex lines might contain more than 60 keypoints. We fix the decoder input as M keypoint queries, where M = 64 keypoints. As consistency in the batch size is not possible due to the diversity in the charts, we add multiple dubious keypoints as (0,0) to the ground truth keypoints so that every chart has exactly 64 keypoints. In some cases, the number of ground truth keypoints is more than 64. In these cases, we only take the first 64 keypoints as our ground truth. In both these cases, we mask them appropriately, with 1 indicating a true ground truth keypoint and 0 indicating a dubious keypoint. This mask is later used during loss calculation so that the model's weights are not affected by the addition of dubious keypoints.

4.1.2 The Loss Function

Unlike 2D pose estimation problems, the precision of keypoint detection is crucial as the predicted keypoint should lie on the correct line and not in the nearby region. Distance-based loss functions such as L1 or L2 fail to capture this intuition. The problem is illustrated in Figure 4. The predicted keypoint (in Red color) is closer to the ground truth keypoint (in Green color) than the other predicted key-

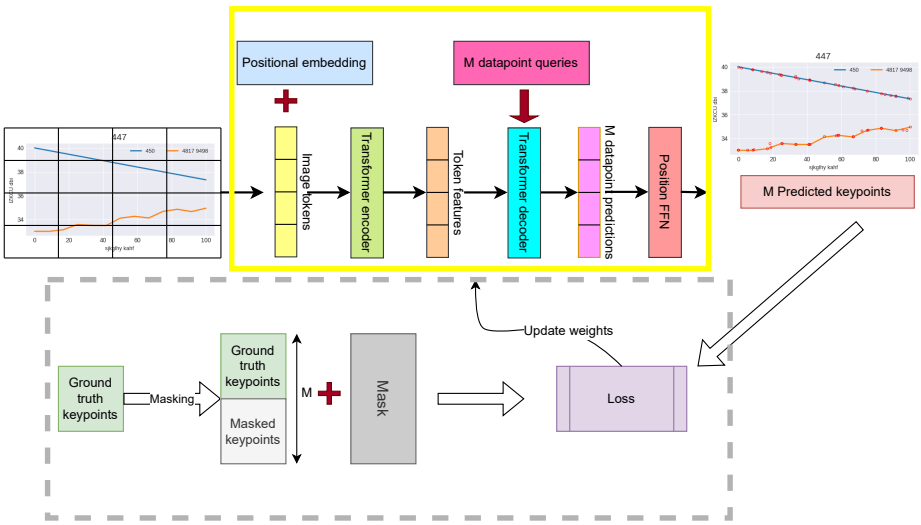


Figure 3: Keypoint detection architecture.

point (in Blue color). However, as keypoint in Blue color lies on the line, it is an ideal candidate than the keypoint in Red color. To capture this intuition, we propose a new loss function and term it as **Angular Similarity Error** ($\mathcal{L}_{\text{A}}$) function.

Before formally defining $\mathcal{L}_{\text{A}}$, we define *anchor points*. Each keypoint in the training line chart has an associated anchor point. The anchor points are determined through a neighborhood pixel search around the ground truth keypoints. Specifically, in our implementation, we define an anchor point as a point that lies on the same line as the ground truth keypoint and is within 5 pixels away from the ground truth keypoint. All the anchor points are computed and stored in the training data. The estimated first derivative using anchor points is then used to calculate the angular separation between the predicted keypoint and the ground truth keypoint.

We now define the $\mathcal{L}_{\text{A}}$ as the angular deviation of the vector joining the predicted keypoint and the ground truth from the first derivative of the line at the ground truth keypoint. The first derivative at the ground truth keypoint is computed using Newton's difference quotient method. The anchor point is used to evaluate the first derivative of the line at the ground truth keypoint. Figure 5 illustrates this idea visually. Consider the ground truth, predicted and anchor point as $\mathbf{Y}$, $\hat{\boldsymbol{Y}}$, and $\mathbf{A}$, respectively. $\mathcal{L}_{\text{A}}$ is formally defined as:

$$\mathcal{L}_{\text{A}} = 1 - cos(\boldsymbol{Y} - \boldsymbol{A}, \boldsymbol{Y} - \hat{\boldsymbol{Y}}) \tag{1}$$

We combine $\mathcal{L}_{\text{A}}$ with standard L1 loss ($\mathcal{L}_{\text{D}}$) to define the overall loss ($\mathcal{L}$) as:

$$\mathcal{L} = \alpha\mathcal{L}_{\text{D}} + (1 - \alpha)\mathcal{L}_{\text{A}} \tag{2}$$

where, α is a hyperparameter.

We propose two variants of our keypoint extraction module, one with $\alpha = 1$ (hereafter referred to as LINEEX$_D$) and other with $\alpha = 0.99$ (hereafter referred to as LINEEX$_{D+A}$). LINEEX$_D$ is a pure L1 loss-based module, whereas LINEEX$_{D+A}$ combines L1 loss with proposed angular similarity error[2].

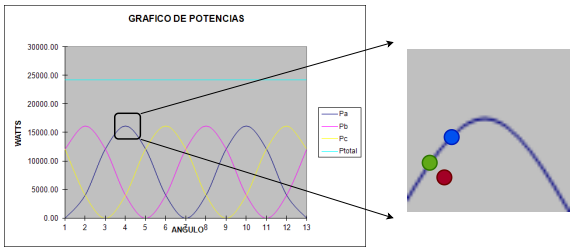


Figure 4: Comparison between predicted keypoints. Ground truth keypoint is denoted by Green color. Predicted keypoints are denoted by Red and Blue colors.

4.2. Chart Element Detection and Text Extraction

The second module identifies axes, legend box, chart title, and their sub-components (described in Section 2). Specifically, it creates a bounding box around the chart title, axes labels, plot area, axes ticks, and legend markers and text. The extracted information from this module helps in several downstream tasks, such as scaling the extracted keypoints from pixel coordinates to the original coordinates and legend mapping. This module is divided into two submodules:

[2]We conduct a binary search to identify the optimal α value.

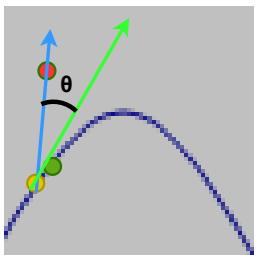


Figure 5: Computing angular similarity error: Anchor point, ground truth keypoint and predicted keypoint are denoted by Yellow, Green and Red color, respectively. Green arrow is a vector that joins the anchor point and the ground truth keypoint. The Blue arrow is a vector joining the ground truth keypoint and the predicted keypoint. The angle between these vectors is denoted by θ.

- **Detection:** We leverage the original DEtection TRansformer (DETR) [5] implementation for predicting bounding boxes around the chart components. DETR is a recently-developed approach that applies the transformer encoder and decoder architecture to object detection and achieves promising performance. To our knowledge, this is the first adoption of the transformer architecture for the problem of chart data extraction. We use the original DETR transformer architecture without making any changes to it, except for adapting its last layer for the following ten classes: (i) Chart title, (ii) X-axis title, (iii) Y-axis title, (iv) Ticks, (v) Plot area, (vi) Inner plot area, (vii) Legend box, (viii) Legend marker, (ix) Legend label, and (x) Legend element. We use a pretrained DETR model and finetune it.
- **Text Extraction:** This sub-module generates chart title, axes labels, and legend texts. We pass the input chart through the EasyOCR [12] tool to read and locate text. EasyOCR outputs the detected text and its corresponding bounding boxes, while the DETR model outputs bounding boxes for each element class. We classify and locate text in the chart by finding the intersection between these bounding boxes. If EasyOCR cannot detect text for a text box detected by DETR, we report that box without any accompanying text.

4.3. Keypoint Grouping, Legend Mapping, and Datapoint Scaling

This module performs three postprocessing tasks, keypoint grouping, legend mapping, and data scaling. We perform keypoint grouping and legend mapping jointly.

- **Keypoint grouping and legend mapping**: We adapt concepts of image similarity to group keypoints and map them to legend markers. We train the Deepranking model [24] by constructing triplets of the keypoint patch and its corresponding positive and negative legend markers. The Deepranking model minimizes hinge loss to group keypoint patches that belong to the same legend marker and differentiate them from keypoint patches that belong to different legend markers. We modify the last linear layer's output size from 4096 to 100 to generate a 100-dimensional embedding vector. Next, we train a simple MLP model to predict the similarity score between the embeddings of a keypoint patch and a legend marker. Figure 6 describes the keypoint grouping and legend mapping module.
  We have the keypoint coordinates and the legend marker bounding boxes during the test phase. We sample 20×40 pixel patches around each detected keypoint and also resize legend marker bounding boxes to 20×40 pixels. Let N be the number of detected legend markers and M be the number of detected keypoints. We send each combination of the keypoint patch and legend marker patch through the above MLP model and obtain a similarity score. This way, we have an N×M matrix of similarity scores between keypoint patches and legend markers. Based on this N×M matrix, we map each keypoint to a unique legend marker. Points at the intersection of two more lines will arbitrarily get mapped to one of the intersecting lines. Every keypoint belonging to the legend marker gets grouped as a detected line, and implicitly, legend mapping is also done.
- **Datapoint Scaling**: The keypoints extracted in Section 4.1 are present in pixel coordinates. In order to transform the pixel coordinates into their corresponding raw data points, we perform a final step of data scaling, which uses the OCR results and tick coordinates obtained in Section 4.2. Algorithm 1 takes in the predicted keypoints, the extracted tick values, and their texts and outputs the scaled datapoint values. Note that X-axis ticks and Y-axis ticks need to be run separately through the algorithm as they may have different scaling.

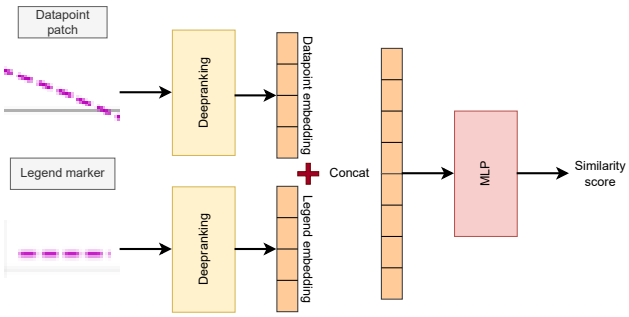


Figure 6: Keypoint grouping and legend mapping module.

Algorithm 1 Datapoint Scaling

Given: list T (extracted tick values), list C (extracted tick coordinates), list dps (unscaled datapoints)
$r\_list \leftarrow []$ ▷ list of ratios corresponding to different ticks pairs
for i in $range(len(C))$ **do**
  for j in $range(len(T))$ **do**
    $r\_list.append(\text{abs}((T[j]-T[i])/(C[j]-C[i])))$
  end for
end for
Find median ratio (r_{med}) and its corresponding indices (med_{idx}) from $coords$
$scaled \leftarrow (dps - C[med_{idx}]) * r_{med} + T[med_{idx}]$

5. Experiments

5.1. Baselines

We list existing systems in Table 1. We only experiment with ChartOCR [15]. ChartOCR is a state-of-the-art system that combines a deep framework and rule-based methods for chart identification and data extraction. We use the trained model available at the project's Github repository[3]. The other closer work, titled *Linear Programming* [14] can not be compared due to the unavailability of source code[4].

5.2. Datasets

We list chart information extraction datasets in Table 2. Due to the unavailability or incomplete ground truth information associated with several datasets, we only conducted experiments on four datasets.

1. **Adobe Synthetic** [6]: The Adobe Synthetic lines test set is used in evaluating the keypoint extraction and the chart element detection modules. The test split comprises of 200 instances.
2. **ExcelChart400k** [15]: The ExcelChart400k train set was used for training and evaluating the keypoint detection module. The train, validation and test split comprises of 116745, 3074 and 3072 instances, respectively.
3. **FigureSeer** [19]: The FigureSeer test set was used in evaluating the chart element detection module. The test set comprises of 1000 instances.
4. **Our Dataset**: Our synthetic dataset was used in training and evaluating the keypoint extraction and the chart element detection modules. The train, validation and test split comprises of 400000, 10000 and 20000 instances, respectively.

[3] https://github.com/soap117/DeepRule

[4] We requested authors, but due to confidentiality clause, the model and the source code cannot be public.

5.3. Evaluation Metrics

Keypoint Extraction: We evaluate the keypoint extraction module by matching predicted keypoints against the ground truth keypoints. Here, we experiment with two variants of similarity metrics both, based on Object Keypoint Similarity score (OKS) [1]. Let $P = [p_1, p_2, p_3, ..., p_M]$ be M predicted keypoints and let $G = [g_1, g_2, g_3, ..., g_m]$ be m ground truth keypoints. For each predicted keypoint p_i, we find the closest ground truth keypoint g_j and the corresponding euclidean distance d_i. Let the length of the chart diagonal be s. Then, OKS for the predicted keypoint p_i is formally defined as:

$$OKS(p_i) = \exp\left(-\frac{{d_i}^2}{2s^2k^2}\right) \tag{3}$$

where k is a breathing zone for the predicted point to be classified as ground truth keypoint [1]. We can assign k such values that can enforce stricter bounds. The first similarity metric variant is stricter than the second one. We describe the two variants below:

- $\mathbf{sim_{str}}$: As the total number of predicted keypoints is significantly higher than the number of ground truth keypoints; we only match one predicted keypoint with one ground truth keypoint. For strict bounds, we use $k = 0.025$ [1]. We define γ as the threshold to compare the OKS value of a predicted keypoint. We set $\gamma = 0.5$. We assign a true positive label to a p_i if $OKS(p_i) > \gamma$ and the closest ground keypoint is not already assigned to some other predicted keypoint. The corresponding ground keypoint is marked as assigned if the assignment is successful.
- $\mathbf{sim_{rel}}$: The predicted keypoint is given a second chance to be labelled as a true positive in the relaxed version. Here, we consider ground truth line segments before assigning a false positive[5] label to the predicted keypoint. Let g_k be the next point in the line in which g_j lies. We define an additional threshold β. A predicted keypoint is assigned as a true positive label after failing the first OKS threshold if the perpendicular distance from p_i to the line segment joining g_j and g_k is less than β. Keeping $\beta = 0.007 \times s$ in the current settings gave the best results.

We calculate recall, precision, and F1 score for each image in the dataset using both the strict and relaxed versions. The final metric scores are averaged over all the images in the test dataset.

One major drawback of our model variants is predicting 64 keypoints irrespective of the number of lines present in the chart image. This can lead to low precision scores. In order to improve precision, we deploy a background detection algorithm. The algorithm detects if the predicted

[5] We consider all predicted keypoints as positive. Their assignment to ground truth keypoints can be treated as True or False.

Algorithm 2 Background Detection

$SecHists$ = CalculateColourHistograms($sections$)
$threshold$ = 0.98
for $hist_1$ in $SecHists$ **do**
  for $hist_2$ in $SecHists$ **do**
    $val \leftarrow$ CompareHist($hist_1$,$hist_2$)
    if $val < threshold$ **then**
      return $False$
    end if
  end for
end for
return $True$

keypoint is on or close to any line in the chart. Let p_i be a predicted keypoint, h and w be the height and width of the image. We take a square patch P centred on p_i with a length of $\max(h, w) \times 0.05$. We further divide P into nine equal smaller square sections. We compare image histograms of the nine sections of the patch using OpenCV's *compareHist* function [2] with the co-relation method. If the image histograms of all nine sections are extremely similar, the predicted keypoint is likely not on or close to a line. Algorithm 2 describes the background detection in detail.

Chart Element Detection Here, we measure the performance of our second module (see Section 4.2 for more details). We use the standard mean Average Precision (mAP) metric for measuring and comparing the performance of different systems. The mAP score is defined as the mean of the Average Precision of each object class in the dataset. The average precision of a class is the weighted mean of the precision values for detecting a bounding box at each confidence threshold. We use the public implementation [16] to evaluate the chart element detection module.

Keypoint Grouping Here, we evaluate our third module (see Section 4.3 for more details) which groups detected keypoints into the lines. We use the modified definition of F1 score [15] to measure keypoint grouping.

Legend Mapping Finally, we evaluate the last module of the pipeline (see Section 4.3 for more details) that maps each detected legend marker to a line that was grouped in the previous step. We use the standard F1 score weighted by the number of lines in each chart.

6. Results and Discussions

Keypoint Extraction Table 3 presents performance scores for keypoint extraction. Among the three systems, LINEEX$_{D+A}$ performed best. ChartOCR outperforms LINEEX on ExcelChart400k dataset. However, it performs poorly on other datasets. We attribute this behaviour to its poor generalization ability (ChartOCR is trained using ExcelChart400k dataset). Simple line charts, such as the charts in the first row in Figure 7, having less number of ground truth keypoints, results in lower precision for LINEEX. However, complex line charts such as sine or cosine graphs (see second row in Figure 7), containing a large number of keypoints yields high precision scores for LINEEX. Empirically, we also find that ChartOCR performs poorly on smooth lines. We validate this finding by conducting an experiment on a 3000 chart images dataset [9]. Each chart comprises multiple smooth lines formed using B-spline and cubic spline interpolated curves. The last three columns in Table 3 validate our empirical finding.

Chart Element Detection Table 4 compares performance scores for chart element detection. LINEEX performs better than ChartOCR on the Adobe Synthetic dataset and our dataset. Both systems perform comparably on the FigureSeer dataset. In contrast to ChartOCR, LINEEX supports the detection of ticks and legend mapping elements (markers and corresponding text). We note that it is difficult to achieve a high mAP score for elements like ticks, legend markers, and legend text due to numerous of them being present in a single chart.

Keypoint Grouping Table 5 compares the keypoint grouping performance. Both systems showcase similar performance. One of the reasons for poor performance on the Adobe dataset is the existence of less popular symbols like cross marks, stars, plus signs, and vertical lines in legend markers. We do not evaluate on ExcelChart400K dataset due to the unavailability of legend marker annotations.

Legend Mapping The F1 score for mapping predicted lines to legend markers was obtained as 0.79.

Ablation Experiments We conduct two ablation experiments. We replace the keypoint extraction transformer encoder in the first experiment with a Resnet50 encoder [11]. In contrast to the Resnet50 encoder, the transformer encoder based LINEEX variant performs 4X better (see Supplementary). In the second experiment, we replace the grouped predicted keypoints with the grouped ground-truth keypoints for legend mapping. We achieve an F1 score of 0.87. This shows that errors introduced in keypoint extraction and grouping stage can lead to low mapping scores.

7. Conclusion

In this paper, we proposed LINEEX for extracting data from scientific line charts using vision transformers. In the future, the proposed work can be expanded into two possible directions: (i) extending to other chart types and (ii) proposal of downstream usecases. The downstream usecases require more precise extraction models, which could be an interesting research direction to explore.

		ExcelChart400K			Adobe Synthetic			Ours			Smooth		
		Recall	Prec	F1	Recall	Prec	F1	Recall	Prec	F1	Recall	Prec	F1
sim_{str}	ChartOCR	**0.85**	**0.98**	**0.90**	0.76	**0.72**	0.71	0.71	**0.90**	0.78	0.36	**0.74**	0.46
	LINEEX$_D$	0.82	0.69	0.70	0.91	0.54	0.64	0.83	0.75	0.76	0.70	0.49	0.56
	LINEEX$_{D+A}$	0.84	0.80	0.78	**0.94**	0.67	**0.74**	**0.86**	0.84	**0.83**	**0.72**	0.52	**0.59**
sim_{rel}	ChartOCR	**0.85**	**0.98**	**0.90**	0.78	0.80	0.76	0.74	**0.97**	0.83	0.38	**0.78**	0.49
	LINEEX$_D$	0.83	0.87	0.83	0.93	0.76	0.81	0.85	0.92	0.87	0.75	0.58	0.64
	LINEEX$_{D+A}$	0.85	0.90	0.85	**0.93**	**0.81**	**0.84**	**0.87**	0.94	**0.89**	**0.77**	0.61	**0.67**

Table 3: Keypoint extraction performance comparison between our proposed method and ChartOCR.

Dataset	Model	Legend Box	Y-axis Title	Chart Title	X-axis Title	Plot Area	Inner Plot Area	Ticks	Legend Marker	Legend Label	Legend Element
Our dataset	ChartOCR	89.07	**99.52**	96.03	85.81	99.69	97.85	-	-	-	-
	LINEEX	**99.97**	83.84	**100.0**	**100.0**	**100.0**	**99.97**	**85.55**	**82.39**	**82.71**	**83.74**
Adobe Synthetic	ChartOCR	-	100.0	81.00	59.50	-	100.0	-	-	-	-
	LINEEX	-	100.0	**99.6**	**100.0**	-	100.0	**57.52**	**74.04**	**88.94**	-
FigureSeer	ChartOCR	-	**97.66**	-	80.96	-	**99.13**	-	-	-	-
	LINEEX	-	70.79	-	**96.28**	-	96.38	-	**54.26**	**63.96**	-

Table 4: Comparison of mAP scores between our proposed model and ChartOCR. As not all models detect all chart components and not each dataset contain all of these chart components, we denote those cells by '-'.

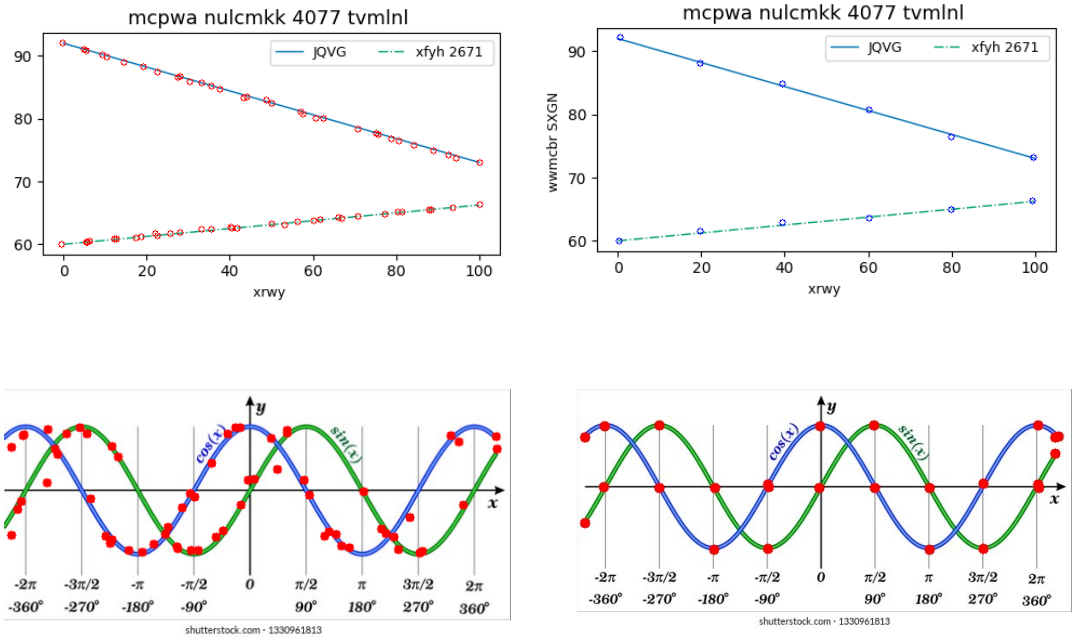


Figure 7: Column 1: LINEEX output, Column 2: ChartOCR output.

	Adobe Synthetic	Our Dataset
ChartOCR	0.54	0.93
LINEEX$_{D+A}$	0.54	0.93

Table 5: F1 scores for keypoint grouping task.

References

[1] Object keypoint similarity. https://cocodataset.org/#keypoints-eval. Accessed: 2022-07-13.

[2] Opencv method for image histogram comparison. https://docs.opencv.org/3.4/d8/dc8/tutorial_histogram_comparison.html. Accessed: 2022-07-04.

[3] Alaaeldin Ali, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. Xcit: Cross-covariance image transformers. *Advances in neural information processing systems*, 34, 2021.

[4] Abhijit Balaji, Thuvaarakkesh Ramanathan, and Venkateshwarlu Sonathi. Chart-text: A fully automated chart image descriptor. *arXiv preprint arXiv:1812.10636*, 2018.

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229, 2020.

[6] Kenny Davila. Competition on harvesting raw tables (chart) 2019 - pubmedcentral, 2019. ICDAR-CHART-2019-PMC `https://tc11.cvc.uab.es/datasets/ICDAR-CHART-2019-PMC_1`.

[7] Kenny Davila. Icpr 2020 competition on harvesting raw tables, 2020. ICPR2020-CHART-Info `https://tc11.cvc.uab.es/datasets/ICPR2020-CHART-Info_1`.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[9] Aalok Gangopadhyay, Prajwal Singh, and Shanmuganathan Raman. Apex-net: Automatic plot extractor network, 2021.

[10] Jinglun Gao, Yin Zhou, and Kenneth E. Barner. View: Visual information extraction widget for improving chart images accessibility. In *19th IEEE International Conference on Image Processing*, pages 2865–2868, 2012.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] JaidedAI. Easyocr, 2022. version 1.4.2 `https://github.com/JaidedAI/EasyOCR`.

[13] Daekyoung Jung, Wonjae Kim, Hyunjoo Song, Jeong-in Hwang, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. Chartsense: Interactive data extraction from chart images. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017.

[14] Hajime Kato, Mitsuru Nakazawa, Hsuan-Kung Yang, Mark Chen, and Björn Stenger. Parsing line chart images using linear programming. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2553–2562, 2022.

[15] Junyu Luo, Zekun Li, Jinpeng Wang, and Chin-Yew Lin. Chartocr: Data extraction from charts images via a deep hybrid framework. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*. The Computer Vision Foundation, January 2021.

[16] Rafael Padilla, Wesley L. Passos, Thadeu L. B. Dias, Sergio L. Netto, and Eduardo A. B. da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3), 2021.

[17] Paschalis Panteleris and Antonis Argyros. Pe-former: Pose estimation transformer. *arXiv preprint arXiv:2112.04981*, 2021.

[18] Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. Revision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, page 393–402, 2011.

[19] Noah Siegel, Zachary Horvitz, Roie Levin, Santosh Divvala, and Ali Farhadi. Figureseer: Parsing result-figures in research papers. *ECCV*, pages 664–680, 2016.

[20] Mayank Singh, Rajdeep Sarkar, Atharva Vyas, Pawan Goyal, Animesh Mukherjee, and Soumen Chakrabarti. Automated early leaderboard generation from comparative tables. In *European Conference on Information Retrieval*, pages 244–257. Springer, 2019.

[21] Sanjay Subramanian, Lucy Lu Wang, Ben Bogin, Sachin Mehta, Madeleine van Zuylen, Sravanthi Parasa, Sameer Singh, Matt Gardner, and Hannaneh Hajishirzi. Medicat: A dataset of medical images, captions, and textual references. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2112–2120, 2020.

[22] Kirill Sviatov, Nadezhda Yarushkina, and Sergey Sukhov. Data extraction of charts with hybrid deep learning model. In *International Conference on Computational Science and Its Applications*, pages 382–393. Springer, 2021.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[24] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1386–1393, 2014.