Bar-JEPA: Extracting Values from Bar Chart with Joint-Embedding Predictive Architecture

Alexander Epple[1*], Poonam Poonam[1*], and Timo Ropinski[1]

Institute of Media Informatics, Visual Computing Group, Ulm University
{alexander.epple, poonam.poonam, timo.ropinski}@uni-ulm.de

Abstract. Bar charts are commonly used in data visualization, and while they are easily understood by humans, it is non-trivial to extract the underlying data computationally. For a machine-learning-based approach, training chart de-rendering models usually requires labeled, real-world data. Labeling data is a time consuming task, which is why annotated data is scarce. Models can learn more efficiently when provided with features of high semantic quality, which a joint-embedding predictive architecture (JEPA) is designed to learn in a self-supervised manner. We present a per-bar, numerical value recovery pipeline for bar charts, where a JEPA encoder is used to produce semantically rich latent features. The decoder model consuming these features is simple and quick to train and outputs the coordinates of ticks and bars, which can be used to recover bar values. The effectiveness of self-supervised finetuning and quality of the extracted features is evident when comparing our model to end-to-end supervised baselines. Code, datasets and checkpoints are available on GitHub.

Keywords: Document analysis · Bar chart value recovery · Document understanding.

1 Introduction

Charts are a popular tool in data visualization, as they can effectively leverage human vision to communicate the underlying data. The raw data is often lost or unavailable when charts are published, as it is easier for humans to read charts than tables [29]. For machines on the other hand, having access to raw data is not only preferable, but necessary for tasks such as data analysis or for accessibility [18]. When extracting data from, or de-rendering charts, it is important for the recovered data to be both numerically and factually accurate, as well as true to the source. This is a challenging task, as there are many graphical and textual components, which need careful, task-specific handling and vary visually [16].

Many ways have been proposed previously to recover data from charts, ranging from semi-automatic [1], to relying on mostly handcrafted features and

* Equal contribution.

methods [32,30] and machine-learning based approaches [25,7,22]. Deep learning methods have achieved state-of-the-art (SoTA) performance on tasks such as chart question answering, chart-to-table, and more, but require vast amounts of labeled data for training [27]. Synthetic data can readily be generated or accumulated [25], but the availability of real-world data, such as the CHART-Info 2024 dataset [9], is much more limited. The problem with synthetic data is, however, that it often does not contain artifacts and struggles to capture the full diversity of real-world data.

In this paper, we propose an approach to deal with real-world, labeled data scarcity by leveraging the self-supervised JEPA [20], which is trained on unlabeled data. We finetune I-JEPA [2], which has been pretrained on ImageNet-1K, on bar charts to produce semantically rich feature maps. For all training tasks, we generate a synthetic bar chart dataset. Then, to test the effectiveness of JEPA for chart de-rendering, we train a set of decoders to extract per-bar numerical values. The decoders first upsample the feature maps and then regress heatmaps for the bars, ticks, and coordinate system origin. They are intentionally designed to be simple and quick to train, as it is not the main focus of the paper. Another problem we address is that aspect ratios of charts can vary significantly. For example, the bar charts in the UB PMC training dataset [10], which consists of manually annotated charts, range all the way from 4.45 to 0.35. Vision transformers (ViT) [11], which I-JEPA uses, require fixed-resolution inputs, so images have to either be resized or padded. To this end, we modify JEPA to accept variable-resolution inputs as proposed in Pix2Struct [21].

To summarize the contributions of this work, we (1) propose a chart data extraction pipeline that uses I-JEPA as a feature extractor, (2) extend I-JEPA to accept variable-resolution inputs and (3) provide a synthetic, 100k bar chart dataset and accompanying generator.

2 Related work

Our work touches two distinct topics: Using JEPA as a feature extractor for some downstream task and chart de-rendering. Our focus, however, is only on extracting values from bar charts.

2.1 Joint-Embedding Predictive Architectures

Since JEPA was proposed by Yann LeCun in 2022 [20], it has been used for various tasks across many modalities to predict representations rather than reconstructing data. Both Fei *et al.* [12] and more recently Tuncay *et al.* [36] use JEPA for audio classification. Fei *et al.* employ a novel curriculum masking strategy, which they find to be crucial for downstream performance and note that the correct strategy depends on the input modality. Tuncay *et al.* on the other hand, note that their training regime required far less pre-training as well as less training data. Riou *et al.* use JEPA for musical stem compatibility estimation, which indicates to what degree an audio file of a single instrument matches

some musical context. They do fall behind their baselines in some tasks, but note the significantly lower amount of training data they had available ($\approx 1\%$ of what the baselines used). These findings support the claim that JEPA is a more data-efficient learning framework than other self-supervised methods.

Point-JEPA by Saito *et al.* [31] learns to predict point clouds, achieves SoTA performance, and the pre-training converges much faster than comparable methods. They point out that JEPA excels in environments where the available data heavily skews towards being unlabeled. Similar to charts, geometry and structure are very important in point clouds, which can be well represented by JEPA.

In the realm of vision models, I-JEPA [2] manages to improve upon Masked Autoencoders (MAE) [15] and Context Autoencoders (CAE) [6] in linear probing at a fraction of the compute. They highlight how I-JEPA converges faster than pixel reconstruction methods such as MAE and CAE, while learning features of a high semantic level. V-JEPA [4] and the more recent V-JEPA 2 [3], which build upon I-JEPA, can solve tasks such as spatio-temporal action detection and image classification without model parameter adaptation. Notably, V-JEPA 2 is particularly effective at tasks that require fine-grained motion understanding, such as pick-and-place on real-world robots. Overall, this shows that pixel-perfect reconstructions are not necessary to achieve good performance on many downstream tasks.

JEPA has also been used in textual contexts, for example TI-JEPA by Vo *et al.* [37] achieves SoTA performance on multimodal sentiment analysis, with the pre-training goal being multimodal alignment of images and text. They manage to bridge the semantic gap between text and image, and reason that this is possible due to the model learning robust and generalizable features. Finally, T-JEPA by Thimonier *et al.* [35] uses JEPA as a pre-training technique for tabular data classification and regression. They manage to match or outperform gradient-boosted decision trees, which are considered to be the go-to method for tabular data.

Clearly, JEPA has been shown to be effective across a wide array of modalities, while being data efficient, learning semantic abstractions and converging fast. To the best of our knowledge it has, however, not yet been applied to charts.

2.2 Chart De-Rendering

Many prior works have applied neural networks to tasks such as chart de-rendering, data recovery and plot question-and-answering (QA). The approaches range from employing convolutional neural networks (CNNs) to transformer architectures and hybrid approaches.

CNNs are a natural fit for charts, as they can effectively exploit the spatial relations of images. For bar charts specifically, object detection-based methods are particularly popular. Liu *et al.* use Faster-RCNN to extract text and values from bar and pie charts [24], while Ma *et al.* rely on ResNet-50 for feature extraction and Cascade R-CNN for element detection [26]. Both networks have multi-stage pipelines, chart specific heads and thus strong inductive biases. The performance suffers when evaluating real-world data compared to synthetic charts, which Liu

et al. attribute to the unavailability of labeled data. Shahira *et al.* propose using Mask R-CNN for bar chart de-rendering, but also note that their approach requires a lot of training data, as well as being slow to train [33]. Several works have instead relied on point-based methods [14,34], where key points are extracted and further processed for value recovery. ChartOCR uses Hourglass Net to classify the chart type and extract key points from which data can be extracted, where the method depends on the chart type [25]. At the time, ChartOCR achieved SoTA performance and it is easy to include additional chart types, which indicates that key point based methods are a good choice for charts. In Zhou *et al.*'s work, bar charts are de-rendered by a CNN in combination with an attention mechanism and long short-term memory (LSTM) [40]. Although the attention mechanism learns to focus on key points, the outputs are vectors containing the coordinates of bar centers and normalized heights. They also note, that their method is unlikely to generalize well to unseen data due to the lack of labeled, real-world data and the diversity thereof.

With the ever growing popularity of transformers, they have also been used in chart de-rendering, being especially popular for chart QA. Pix2Struct [21], MatCha [23] and DePlot [22] all build upon each other and use ViT [11] encoders and text decoders. The models achieve SoTA results on plot QA tasks and significantly outperform the CNN-based ChartOCR [25]. Pix2Struct introduces variable-resolution inputs, as maintaining the original aspect ratio helps to improve model performance. There are many more examples of transformer-based chart-to-table models [7,28,27], which also show promising results for chart de-rendering. Key point detection is also possible with transformers as shown by Xue *et al.* in ChartDETR [39], building upon DETR [5], which is an end-to-end object detector. Their method is straightforward and, while being conceptually similar, improves upon ChartOCR in terms of robustness and performance. These findings indicate that transformers can often outperform CNN-based approaches and therefore, are a good choice for the task of chart de-rendering.

It is worth pointing out, however, that even though the training objectives are diverse, all prior work, as opposed to JEPA [2], fall into the field of supervised learning. To our knowledge, little to no work has been done on self-supervised pretraining in chart de-rendering. Existing methods focus on optimizing end-to-end performance and representation quality is rarely taken into account.

3 Method

This section covers the dataset used for training and the simplifying assumptions we make to scope the research area. As our goal is to study representation quality, we limit complexity and ensure a controlled setting. An overview of I-JEPA follows, as well as an explanation of our modifications to the framework and the self-supervised pretraining. The decoders and their supervised training regime are outlined next, before finally the numerical value extraction is detailed.

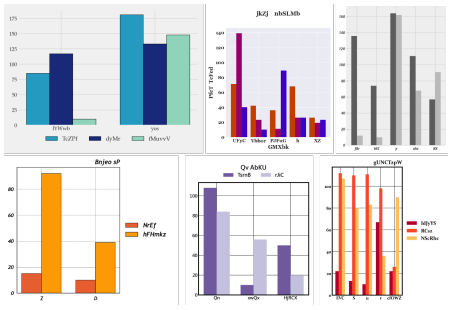

Fig. 1. Examples of generated bar charts as used for encoder finetuning and decoder training. Charts are generated according to the parameters in Table 5.

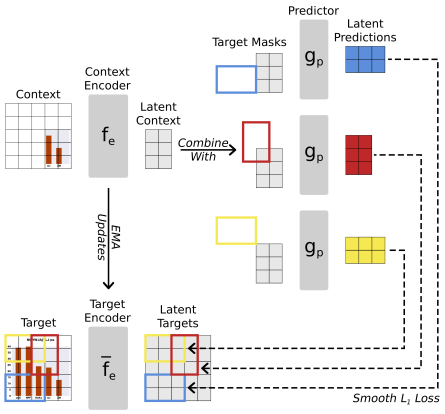

Fig. 2. Overview of the I-JEPA training procedure. The context encoder is used after training concludes, predictor and target encoder are discarded.

3.1 Synthetic data generator and datasets

Bar charts come in considerable visual variety and can contain a lot of visual clutter, so in order to narrow the amount of variables, we make some assumptions for the dataset we generate for training. We only consider vertical bar charts and the bars may not be stacked, contain error bars or 3D effects. This ensures the charts have visual diversity but are kept simple. The parameters used to generate our datasets are listed in Table 5 and we use matplotlib [17] to generate charts. We extend the work of Zhou *et al.* [40] by, for example, varying which random distribution is used to generate values and by outputting a more exhaustive annotation file. The annotations contain precise bounding boxes and values of all chart elements. Some examples of the charts we generate can be seen in Figure 1. For the I-JEPA finetuning objective, we generate $100k$ charts, all of which are used for training, as there is no validation or test step. The dataset for the decoder pretraining consists of $17k$ synthetic charts. Additionally, we use the UB PMC dataset [9] (specifically, ICPR CHART-Infographics 2022) for decoder finetuning. We can only use 1316 of the vertical bar charts, as the remaining data does not have the required labels. Both datasets are split into 80 : 20% training : validation sets.

3.2 I-JEPA encoder

Our encoding pipeline starts with variable-resolution patch extraction, following the implementation from Pix2Struct [21]. Given a maximum number N of patches, for each input image of size (w, h) the amount of feasible rows and columns (R, C) are computed according to Equation 1 and Equation 2. Then,

the image is resized to (W, H) and a sequence of $M \leq N$ non-overlapping patches of size p are extracted. An overview of the process can be seen in Figure 3.

$$R = \max\left(1,\ \min\left(N,\ \left\lfloor \sqrt{N}\sqrt{\frac{h}{w}} \right\rfloor\right)\right), \qquad H = R\,p \tag{1}$$

$$C = \max\left(1,\ \min\left(N,\ \left\lfloor \sqrt{N}\sqrt{\frac{w}{h}} \right\rfloor\right)\right), \qquad W = C\,p \tag{2}$$

The training objective of I-JEPA is similar to that of MAEs [15], but the key differences are that it is (1) non-generative and (2) predicts in latent as opposed to pixel space. I-JEPA consists of three ViTs [11]: The context encoder f_e, target encoder $\bar{f}_e$ and predictor g_p. The context and target encoders are structurally identical, while the predictor is narrower and shallower. Given a latent context block from an encoded image and a target mask, the predictor learns to reproduce the latent patches of the target block in the same image. The target and context blocks are obtained using multi-block masking [2]. For the exact parameters, see Table 6 and refer to Figure 2 for an visual overview of the model.

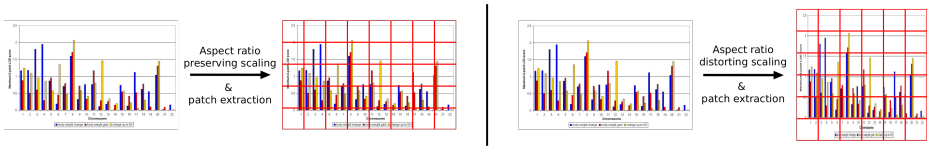

Fig. 3. Aspect-ratio preserving patch extraction ensures at most N patches are obtained after resizing an image. The operation guarantees the new size to be close to the original aspect-ratio as much as possible. The chart is an example from [10].

Targets. For each image, target blocks are obtained by first sampling i blocks B_i from the M input patches. The size and aspect-ratio of each B_i is random and the blocks can overlap. They are encoded by $\bar{f}_e$ to yield sequences $\bar{s}_i$, where each sequence $\bar{s}_i$ contains up to j latent patches. The target encoder is frozen and receives no gradient flow during backpropagation, which prevents the encoders from collapsing. Without this, the encoders' output could simply always be a constant value, which makes predictions trivial and the representations meaningless.

Context. There is usually only one context block B_x of random size and aspect-ratio, where any patch regions that overlap with the target blocks B_i are removed. The remaining patches in B_x are encoded by f_e into a sequence s_x of latent patches. The context encoder receives gradients during backpropagation and updates the target encoder $\bar{f}_e$ via exponential moving average.

Predictions. Instead of predicting all target blocks at once, the predictor g_p is invoked for each target B_i individually. It receives as input (1) the context patches s_x and (2) learned mask tokens t_i with added positional embeddings. The mask tokens t_i tell the predictor where the target block B_i is and enables it to generate the sequence $\hat{s_i}$ of latent patch predictions.

Loss. After the target and prediction sequences have been obtained, the loss $\mathcal{L}$ is calculated according to Equation 3 by accumulating the smooth L1 loss (Equation 4) of the targets $\bar{s_i}$ and predictions $\hat{s_i}$ for each block B_i.

$$\mathcal{L} = \sum_i \text{smoothL1}(\bar{s}_i, \hat{s}_i) \tag{3}$$

$$\text{smoothL1}(x, y) = \begin{cases} \frac{1}{2}(x-y)^2 & \text{if } |x-y| < 1 \\ |x-y| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{4}$$

Training. We use the ViT-H ImageNet-1K checkpoint from the original I-JEPA paper [2] and finetune for 50 epochs. The model is trained using the AdamW optimizer, with cosine weight decay between 0.02 and 0.04 and a cosine learning rate schedule from $5.0e-5$ to $1.0e-6$ with a 6-epoch warm-up starting at $5.0e-6$. We train both variable-resolution and fixed-resolution models on the aforementioned dataset of $100k$ generated charts using an effective batch size of 40 on two NVIDIA RTX A6000 GPUs. Similarly to Thimonier *et al.* [35], we observe that the loss starts at a collapsed equilibrium before rising and converging. This is also the case for I-JEPA pretraining [2] and expected behavior.

3.3 Key point extractor

Our extraction pipeline, which can be seen in Figure 4, is intentionally simple and starts with the features extracted by the frozen encoder. These features with latent channel dimension size d and patch size p are of shape $F_{jepa} \in \mathbb{R}^{d \times \frac{H}{p} \times \frac{W}{p}}$. The decoder consumes these features directly and is the only part that is trained.

Decoders. To keep the decoders lightweight, we use the simple and classic decoders described in ViTPose [38] but with $N_k = 32$ heatmap channels. Conceptually, this is supposed to encourage the model to put one key point into each channel during training. We enforce that the first channel contains the coordinate system origin, channels $1-15$ ticks, and the rest is reserved for bars. This is accomplished by the ground truth heatmaps being in the given order. Ticks are to be output from the bottom to the top and bars from left to right.

The simple decoder consists of a ReLU activation, followed by $4\times$ bilinear upsampling and a 3×3 convolution. For the classic decoder, there are two blocks each consisting of a deconvolution, batch normalization and ReLU activation. The two blocks are followed by a single 1×1 convolution. The final convolutions shape the feature maps to $F_{kp} \in \mathbb{R}^{N_k \times \frac{H}{4} \times \frac{W}{4}}$ by reducing the channels down to N_k key point maps.

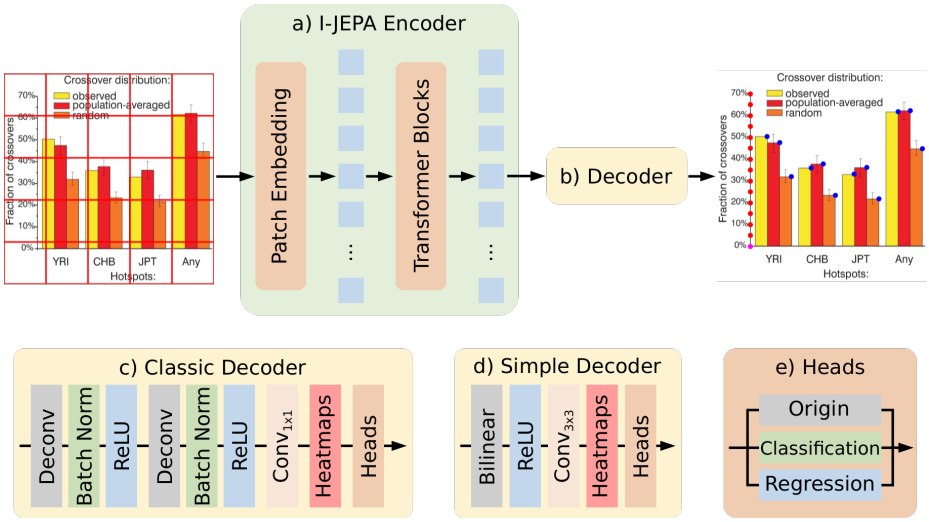

Fig. 4. The decoder pipeline consists of the frozen, pretrained I-JEPA encoder (a) and a decoder (b), which produces key point heatmaps. The heatmaps are consumed by the output heads (e), which in turn generate a combined heatmap. There are two possible configurations, a simple (d) and a slightly more sophisticated (c) decoder model. The chart is an example from [10].

Heatmap heads. The key point maps are consumed by three heads, which produce an origin, classification and regression map and mirror the method of Soto *et al.* [34]. The origin map $H_{org} \in \mathbb{R}^{1 \times \frac{H}{4} \times \frac{W}{4}}$ helps guide learning early on and predicts the location of the coordinate system origin. The channels of the classification map $H_{cls} \in \mathbb{R}^{3 \times \frac{H}{4} \times \frac{W}{4}}$ correspond to the likelihood of a pixel being background, bar or tick. The regression map $H_{reg} \in \mathbb{R}^{2 \times \frac{H}{4} \times \frac{W}{4}}$ accounts for point offsets accrued by the low resolution. An example of the outputs can be seen in Figure 5.

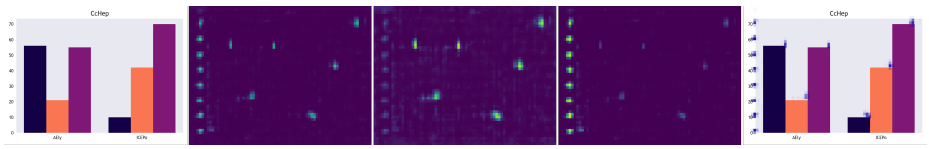

Fig. 5. An example of the heatmaps produced by the decoder. From left to right: Input image, (inverted) background map, top right bar corners, ticks and input image overlaid with all three heatmaps.

Heatmap loss. We use a composite loss function, consisting of separate losses for each of the three output maps and an extra loss for the key point channels. Since the decoders are trained with supervision, ground truth maps are used to compute the loss (Equation 5). We find through experimentation that the model is not able to output individual key point heatmaps into the key point channels directly, but the additional loss does significantly help the classification head to separate ticks from bars.

$$\mathcal{L}_{total} = \lambda_1 \mathcal{L}_{org} + \lambda_2 \mathcal{L}_{cls} + \lambda_3 \mathcal{L}_{reg} + \lambda_4 \mathcal{L}_{kp} \tag{5}$$

$\mathcal{L}_{org}$*:* The origin loss is computed as the MSE between a Gaussian target heatmap ($\sigma = 1.0$) and the output H_{org}. The origin loss is weighted equally to the classification loss ($\lambda_1 = 1.0$) and, on average, contributes $\approx 25\%$ in relation to $\mathcal{L}_{cls}$ during training.

$\mathcal{L}_{cls}$*:* For classification we use cross entropy on H_{cls}, with the background class being weighted 0.05, whereas bars and ticks are weighted equally at 1.0 due to class imbalance. Classification being most important for value recovery is weighted $\lambda_2 = 1.0$.

$\mathcal{L}_{reg}$*:* Regression uses mean squared error (MSE) and only tick and bar pixels in H_{reg} receive gradients; all other pixels are masked out. This loss is more sparse, so we set $\lambda_3 = 4.0$, but during training $\mathcal{L}_{reg}$ is, on average, only $\approx 33\%$ of $\mathcal{L}_{cls}$.

$\mathcal{L}_{kp}$*:* The heatmap point loss operates on the key point maps F_{kp} and is supposed to encourage the decoder to output key points earlier, so the heads can combine them into the final outputs. As described earlier, each channel corresponds to one specific key point and its loss contribution is calculated the same as $\mathcal{L}_{org}$. Only channels with an associated ground truth key point receive gradients and the loss is normalized by the number of active channels. We weight $\lambda_4 = 4.0$, although $\mathcal{L}_{kp}$ and $\mathcal{L}_{cls}$ contribute equally to the overall loss.

Training The decoders are trained for 50 epochs on our generated dataset consisting of $17k$ charts. We use a cosine learning rate schedule from $1.0e-3$ to $1.0e-5$ and a 3 epoch warm-up starting at $2.0e-4$. Optimization is handled by AdamW with a cosine weight decay schedule between 0.04 and 0.1. The training batch size is 1360 on a single NVIDIA RTX A6000 GPU. After pretraining concludes, we finetune for 30 more epochs on the UB PMC training dataset [9]. Decoder finetuning follows the same regime as in section 3.2, with a batch size of 263 on a single NVIDIA RTX A6000 GPU, the batch size being tailored to divide the small number of samples evenly.

3.4 Value recovery

To translate the heatmaps into useful data, we follow the method of Soto *et al.* [34] with some modifications: First, the bar and tick maps from H_{cls} are

background masked at a threshold of 0.75 and surviving candidates from each map with confidence $c \geq 0.75$ are transformed to normalized image coordinates. The offsets from H_{reg} are added to allow for sub-pixel accuracy. Next, confidence-based non-maximum suppression (NMS) is applied to the candidates, where the radius r_{nms} can account for variable-resolution inputs and is the equivalent distance of 1.5 pixels in heatmap size (Equation 6). Afterwards, the remaining coordinates are the predicted bar and tick positions in normalized image space.

$$r_{\text{nms}} = \frac{1.5 \cdot p}{4 \cdot \sqrt{HW}} \tag{6}$$

The next step consists of converting tick labels to numerical data. As optical character recognition (OCR) is not the focus of this work, we use PaddleOCR [8] for this task, specifically the *latin_PP-OCRv5_mobile_rec* text recognition model with the language set to English. For all detections containing numerical text (determined using regular expressions), the extracted values and bounding box centers are matched with the predicted tick positions using the Hungarian algorithm [19]. A match is only considered to be valid if the distance d between label and tick is $d \leq 5.0 \times r_{\text{nms}}$ to avoid numerical text that does not stem from labels being matched with ticks. Similar to Zhou *et al.* [40], we fit a line for matched value and tick coordinates with RANSAC regression [13]. The resulting function can then be used to predict values from bar coordinates.

4 Experiments

This section outlines the metrics used for evaluation, shows the effectiveness of finetuning the encoder on domain-specific data, and explains the influence of variable-resolution inputs, as well as the decoder choice has on performance. Finally, a short comparison to prior work highlights the strengths and shortcomings of our approach.

Table 1. Quantitative results of four models evaluated on two datasets. The variable- and fixed-resolution models, as well as the vanilla model, use the *classic* decoder from Figure 4 and the *simple* decoder receives variable-resolution inputs.

Dataset	Model	Bar F1	Tick F1	Acc. ($\varepsilon = 0.05$)	Acc. ($\varepsilon = 0.02$)
UB PMC [9]	Variable Resolution	0.740	0.827	0.450	0.341
	Fixed Resolution	0.716	0.812	0.422	0.292
	Vanilla	0.161	0.024	0.000	0.000
	Simple Decoder	0.085	0.178	0.009	0.005
Synthetic Charts	Variable Resolution	0.956	0.940	0.778	0.629
	Fixed Resolution	0.882	0.892	0.654	0.498
	Vanilla	0.389	0.182	0.056	0.051
	Simple Decoder	0.124	0.253	0.014	0.006

4.1 Metrics and test datasets

All results are evaluated on two datasets, namely 100 synthetic charts that we generate specifically for testing and split 4 of the UB PMC [9] test set (ICPR CHART-Infographics 2022). There are only 139 vertical bar charts in this split that have the labels required for evaluation. The results we report are the F1 score for bar and tick detections, as well as the value recovery accuracy.

To calculate the F1 score, we use the Hungarian algorithm [19] with the precision threshold $r_{\text{f1}} = r_{\text{nms}}$ (Equation 6) to match predicted and ground truth coordinates. When evaluating fixed-resolution inputs, the heatmaps are 64×64 pixels, so the error of a correct prediction is at most $\varepsilon \approx 2.3\%$. For consistency, we use the same threshold for both variable and fixed-resolution inputs, even though this could, in some cases, mean a fixed-resolution detection would not be considered valid in a variable-resolution setup and vice-versa.

Value recovery is measured using accuracy. We specifically use the same criterion as Zhou *et al.* [40] as described in Equation 7.

$$\frac{|h_g - h_p|}{h_g} \leq \varepsilon \tag{7}$$

Here, h_g is the ground truth value, h_p the predicted value and ε the strictness. For our results, we use the same relaxed ($\varepsilon = 0.05$) and hard ($\varepsilon = 0.02$) accuracy thresholds. We also account for missing and additional predictions, but for simplicity no OCR correction is applied.

4.2 Ablation study and analysis

In order to evaluate the influence of our contributions and implementation choices, we train four different models as described in Table 2 according to the schedule outlined in section 3.3.

Table 2. The models used for the ablation studies. All encoders originate from the ViT-H ImageNet-1K checkpoint. For aspect ratio preserving (ARP) models the encoder was finetuned with ARP enabled. The decoder types are described in Figure 4.

Model	Aspect Ratio Preserving	Decoder	Encoder
Variable Resolution	✓	Classic	Finetuned
Fixed Resolution	×	Classic	Finetuned
Vanilla	×	Classic	ViT-H ImageNet-1K
Simple Decoder	✓	Simple	Finetuned

The influence of encoder finetuning. Arguably, the most important question was whether or not self-supervised encoder finetuning on charts has an impact on downstream performance. As is evident from the results in Table 1, this

very much appears to be the case: On synthetic charts, the F1 scores of the fixed-resolution, finetuned encoder are more than double that of the vanilla encoder. In case of value recovery, the difference is even more pronounced. This likely is the result of compounding errors, as it depends on both bars and ticks being accurately detected. When evaluating the models on real-world data, the vanilla encoder completely fails at recovering any values, whereas the finetuned encoder has a relaxed accuracy of 42%. With the only difference between these two models being the additional training step, self-supervised finetuning on charts clearly is able to yield a significantly more capable encoder for downstream tasks.

The influence of variable-resolution inputs. The second question we want to answer concerns the impact of variable-resolution inputs. In Pix2Struct [21], this simple change resulted in $\approx 5\%$ accuracy uplift in their warm-up stage. We see a similar benefit in value recovery accuracy, the difference being $\approx 5\%$ for real world data and $\approx 13\%$ for synthetic charts at the hard accuracy threshold. Considering this modification comes at negligible cost during training and inference time, the integration is well worth it.

The influence of decoder choice. Finally, we find that the results of ViTPose [38] do not transfer to our method. Whereas they see almost no performance difference between the two decoders for human pose estimation, the simple decoder performs significantly worse for our task. It fails to recover values from both synthetic and real-world charts, with the accuracy hovering at $\approx 1\%$. When evaluating the heatmaps directly, it seems to be unable to focus in on individual key points and instead outputs large splats. Our hypothesis is that the decoder is over-constrained and does not have enough capacity for the task at hand. Our network is not trained end-to-end, which could possibly explain the different findings: If the encoder in ViTPose does a lot of the heavy-lifting, the decoder choice would matter a lot less.

4.3 Comparison to other papers

It is somewhat difficult to compare our results to prior work directly, as our outputs are limited to bar and tick coordinates. Nevertheless, we can draw a comparison to the works of Soto *et al.* [34] and Zhou *et al.* [40]. We can only compare results conceptually on qualitatively similar data due to the unavailability of the datasets they used in testing. As can be seen in Table 4, our method performs similarly well on synthetic data at the relaxed threshold ($\varepsilon = 0.05$). Since our chart generator builds on theirs, the comparison on synthetic data is fair. While our method does fall behind on real-world data, most of the charts in our test set do not fulfill the criteria Zhou *et al.* assume (e.g. include error bars / stacked bars). To compare our work to Soto *et al.*, we use the mean tick and bar F1 scores over the entire test set. Their synthetic data is, again, very comparable to ours and the real-world data they curated, just like UB PMC [9], consists of charts from PubMedCentral. The results in Table 3 show that, while

Table 3. The mean F1 score of our model and Soto *et al*'s [34] trained on synthetic data. The test datasets, while not being the same, are very similar, and our precision threshold is also slightly stricter (2.3% vs. 2.7%).

Method	Data	F1 Score
Ours	Synthetic	0.948
Soto *et al.* [34]	Synthetic	**0.981**
Ours	Real World	**0.774**
Soto *et al.* [34]	Real World	0.555

Table 4. Value recovery accuracy at the relaxed threshold ($\varepsilon = 0.05$) of our model and Zhou *et al.*'s [40] without OCR correction. The test datasets differ. Our real-world data is more varied, whereas the synthetic data is nearly identical.

Method	Dataset	Accuracy
Ours	Synthetic	79%
Zhou *et al.* [40]	Synthetic	**82%**
Ours	Real World	45%
Zhou *et al.* [40]	Real World	**71%**

our model performs slightly worse on synthetic data (≈ 0.03 difference), we can improve upon their results when it comes to real-world data by ≈ 0.22. It should be noted, however, that we do not have many trainable parameters in the value recovery pipeline, as the bulk of the parameters live in the frozen encoder. Additionally, Zhou *et al.* [40] train on $30k$ charts for 300 epochs and Soto *et al.* 1750 epochs on $5k$ charts, which is significantly more than it takes for our model to converge (see section 3.3). This suggests that self-supervised finetuning, as well as the semantic quality of the extracted features are the driving factor behind our results.

5 Discussion and Conclusion

In this study, we propose utilizing I-JEPA, a self-supervised learning model, as a feature extractor for chart de-rendering. To our knowledge, this is the first time that JEPA has been used in the context of chart understanding, a traditionally supervision-heavy task. We were able to show that finetuning the feature extractor is highly beneficial for downstream performance and our results suggest that the features are indeed of high semantic quality. This is supported by the fact that our lightweight value recovery model converges quickly during training and does not require a large corpus to achieve good results. We also introduce variable resolution inputs to I-JEPA, which yields modest performance improvements with negligible overhead.

Our work does, however, have some limitations: (1) Our model is rather simplistic and can only recover values of one type of chart, (2) it does not achieve SoTA results at this task and (3) it may prove difficult to integrate our feature extractor into multimodal language models (e.g. for chart QA). Although we are not trying to compete with SoTA models, it would be beneficial to build a more competent decoder in future work to show the full potential of our method. Our encoder is frozen during training and I-JEPA operates exclusively in latent space, so aligning a text-based model could be challenging. Considering many SoTA models are to some extent based on large language models [22,7,28,27], the ability to train end-to-end might be beneficial.

There are several possible avenues for future work. Implementing a more powerful decoder, such as the transformer-based ChartDETR [39] could result in better de-rendering capabilities. Including a large language model in the pipeline or replacing the decoder with a large language model would allow testing if our method is also suited for chart QA. Taking ChartGemma [28] as an example, it may be possible to use I-JEPA as the vision encoder and align it with the language model by training an embedding layer. Finally, scaling the model to more chart types and finetuning on a more diverse corpus would be a logical next step, as it is unclear how much data is required for finetuning. Judging the quality of the extracted features directly is a hard task by itself and it is entirely possible that more data and longer training would yield even stronger features.

Acknowledgments. We acknowledge the EuroHPC Joint Undertaking for awarding this project access to the EuroHPC supercomputer LEONARDO, hosted by CINECA (Italy) and the LEONARDO consortium through an EuroHPC Development Access call. The authors acknowledge support by the state of Baden-Württemberg through bwHPC.

References

1. PlotDigitizer: Extract Data from Graph Image Online — plotdigitizer.com. https://plotdigitizer.com, [Accessed 19-01-2026]
2. Assran, M., Duval, Q., Misra, I., Bojanowski, P., Vincent, P., Rabbat, M., LeCun, Y., Ballas, N.: Self-supervised learning from images with a joint-embedding predictive architecture. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 15619–15629 (2023)
3. Assran, M., Bardes, A., Fan, D., Garrido, Q., Howes, R., Muckley, M., Rizvi, A., Roberts, C., Sinha, K., Zholus, A., et al.: V-jepa 2: Self-supervised video models enable understanding, prediction and planning. arXiv preprint arXiv:2506.09985 (2025)
4. Bardes, A., Garrido, Q., Ponce, J., Chen, X., Rabbat, M., LeCun, Y., Assran, M., Ballas, N.: V-JEPA: Latent video prediction for visual representation learning (2024), https://openreview.net/forum?id=WFYbBOEOtv
5. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: European conference on computer vision. pp. 213–229. Springer (2020)
6. Chen, X., Ding, M., Wang, X., Xin, Y., Mo, S., Wang, Y., Han, S., Luo, P., Zeng, G., Wang, J.: Context autoencoder for self-supervised representation learning. International Journal of Computer Vision **132**(1), 208–223 (2024)
7. Cheng, Z.Q., Dai, Q., Hauptmann, A.G.: Chartreader: A unified framework for chart derendering and comprehension without heuristic rules. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 22202–22213 (2023)
8. Cui, C., Sun, T., Lin, M., Gao, T., Zhang, Y., Liu, J., Wang, X., Zhang, Z., Zhou, C., Liu, H., et al.: Paddleocr 3.0 technical report. arXiv preprint arXiv:2507.05595 (2025)

9. Davila, K., Lazarus, R., Xu, F., Rodríguez Alcántara, N., Setlur, S., Govindaraju, V., Mondal, A., Jawahar, C.: Chart-info 2024: A dataset for chart analysis and recognition. In: International Conference on Pattern Recognition. pp. 297–315. Springer (2024)
10. Davila, K., Xu, F., Ahmed, S., Mendoza, D.A., Setlur, S., Govindaraju, V.: Icpr 2022: Challenge on harvesting raw tables from infographics (chart-infographics). In: 2022 26th International Conference on Pattern Recognition (ICPR). pp. 4995–5001. IEEE (2022)
11. Dosovitskiy, A.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
12. Fei, Z., Fan, M., Huang, J.: A-jepa: Joint-embedding predictive architecture can listen. ArXiv **abs/2311.15830** (2023), https://api.semanticscholar.org/CorpusID:265456289
13. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM **24**(6), 381–395 (1981)
14. Hassan, M.Y., Singh, M., et al.: Lineex: Data extraction from scientific line charts. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 6213–6221 (2023)
15. He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked autoencoders are scalable vision learners. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 16000–16009 (2022)
16. Huang, W., Tan, C.L.: A system for understanding imaged infographics and its applications. In: Proceedings of the 2007 ACM symposium on Document engineering. pp. 9–18 (2007)
17. Hunter, J.D.: Matplotlib: A 2d graphics environment. Computing in Science & Engineering **9**(3), 90–95 (2007). https://doi.org/10.1109/MCSE.2007.55
18. Joshi, I.: Unblind the charts: Towards making interactive charts accessible in android applications. ArXiv **abs/2109.12442** (2021), https://api.semanticscholar.org/CorpusId:237940794
19. Kuhn, H.W.: The hungarian method for the assignment problem. Naval research logistics quarterly **2**(1-2), 83–97 (1955)
20. LeCun, Y.: A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. Open Review **62**(1), 1–62 (2022)
21. Lee, K., Joshi, M., Turc, I.R., Hu, H., Liu, F., Eisenschlos, J.M., Khandelwal, U., Shaw, P., Chang, M.W., Toutanova, K.: Pix2struct: Screenshot parsing as pretraining for visual language understanding. In: International Conference on Machine Learning. pp. 18893–18912. PMLR (2023)
22. Liu, F., Eisenschlos, J., Piccinno, F., Krichene, S., Pang, C., Lee, K., Joshi, M., Chen, W., Collier, N., Altun, Y.: Deplot: One-shot visual language reasoning by plot-to-table translation. In: Findings of the Association for Computational Linguistics: ACL 2023. pp. 10381–10399 (2023)
23. Liu, F., Piccinno, F., Krichene, S., Pang, C., Lee, K., Joshi, M., Altun, Y., Collier, N., Eisenschlos, J.: Matcha: Enhancing visual language pretraining with math reasoning and chart derendering. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 12756–12770 (2023)
24. Liu, X., Klabjan, D., NBless, P.: Data extraction from charts via single deep neural network. arXiv preprint arXiv:1906.11906 (2019)

25. Luo, J., Li, Z., Wang, J., Lin, C.Y.: Chartocr: Data extraction from charts images via a deep hybrid framework. In: Proceedings of the IEEE/CVF winter conference on applications of computer vision. pp. 1917–1925 (2021)
26. Ma, W., Zhang, H., Yan, S., Yao, G., Huang, Y., Li, H., Wu, Y., Jin, L.: Towards an efficient framework for data extraction from chart images. In: International Conference on Document Analysis and Recognition. pp. 583–597. Springer (2021)
27. Masry, A., Kavehzadeh, P., Do, X.L., Hoque, E., Joty, S.: Unichart: A universal vision-language pretrained model for chart comprehension and reasoning. arXiv preprint arXiv:2305.14761 (2023)
28. Masry, A., Thakkar, M., Bajaj, A., Kartha, A., Hoque, E., Joty, S.: Chartgemma: Visual instruction-tuning for chart reasoning in the wild. In: Proceedings of the 31st International Conference on Computational Linguistics: Industry Track. pp. 625–643 (2025)
29. Meyer, J., Shamo, M., Gopher, D.: Information structure and the relative efficacy of tables and graphs. Human Factors: The Journal of Human Factors and Ergonomics Society **41**, 570 – 587 (1999), https://doi.org/10.1518/001872099779656707
30. Mishchenko, A., Vassilieva, N.: Chart image understanding and numerical data extraction. In: 2011 sixth international conference on digital information management. pp. 115–120. IEEE (2011)
31. Saito, A., Poovvancheri, J.: Point-jepa: A joint embedding predictive architecture for self-supervised learning on point cloud. 2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) pp. 7348–7357 (2024), https://api.semanticscholar.org/CorpusID:269362564
32. Savva, M., Kong, N., Chhajta, A., Fei-Fei, L., Agrawala, M., Heer, J.: Revision: Automated classification, analysis and redesign of chart images. In: Proceedings of the 24th annual ACM symposium on User interface software and technology. pp. 393–402 (2011)
33. Shahira, K., Joshi, P., Lijiya, A.: Data extraction and question answering on chart images towards accessibility and data interpretation. IEEE Open Journal of the Computer Society **4**, 314–325 (2023)
34. Soto, C., Yoo, S.: An extensible point-based method for data chart value detection. arXiv preprint arXiv:2308.11788 (2023)
35. Thimonier, H., Costa, J.L.D.M., Popineau, F., Rimmel, A., Doan, B.L.: T-jepa: Augmentation-free self-supervised learning for tabular data. arXiv preprint arXiv:2410.05016 (2024)
36. Tuncay, L., Labb'e, E., Benetos, E., Pellegrini, T.: Audio-jepa: Joint-embedding predictive architecture for audio representation learning. ArXiv **abs/2507.02915** (2025), https://api.semanticscholar.org/CorpusID:280151263
37. Vo, K.H., Nguyen, D.P., Nguyen, T.T., Quan, T.T.: Ti-jepa: An innovative energy-based joint embedding strategy for text-image multimodal systems. In: International Symposium on Information and Communication Technology. pp. 141–154. Springer (2024)
38. Xu, Y., Zhang, J., Zhang, Q., Tao, D.: Vitpose: Simple vision transformer baselines for human pose estimation. Advances in neural information processing systems **35**, 38571–38584 (2022)
39. Xue, W., Chen, D., Yu, B., Chen, Y., Zhou, S., Peng, W.: Chartdetr: A multi-shape detection network for visual chart recognition. arXiv preprint arXiv:2308.07743 (2023)
40. Zhou, F., Zhao, Y., Chen, W., Tan, Y., Xu, Y., Chen, Y., Liu, C., Zhao, Y.: Reverse-engineering bar charts using neural networks. Journal of Visualization **24**(2), 419–435 (2021)

A Parameters

Table 5. Parameters used to generate our datasets. The colors are selected to be rich in contrast, to avoid white-on-white charts. Fonts have to support the roman alphabet and we filter out specialty fonts (e.g. math, pictographs).

Parameter	Details
Chart size	600–1600 px (width & height)
Bar series	2–5
Bars per series	1–3
Bar values	10–200
Bar width & spacing	Random in accordance with size
Bar colors	Randomly sampled from color maps
Font & font size	Randomly selected from roman fonts
Axis label length	5–15 characters
Ticks label length	1–5 characters
Title length	5–15 characters
Title location	left, center, right
Legend length	3–6 characters
Legend location	top, bottom, right

Table 6. I-JEPA finetuning parameters. We use the ViT-H preset for training and the default multiblock sampling strategy, with the same settings as used during pretraining. For both variable- and fixed-resolution inputs we allow at most 256 patches of size 14.

Parameter	Details
Context Block Count	1
Context Block Size	85–100%
Predictor Block Aspect Ratio	0.75–1.5
Predictor Block Count	4
Predictor Block Size	15–20%
Maximum Patch Count	256