Crafting Scalable Methodologies for Creating Domain-Representative Chart Derendering Datasets

Kieran Swenson

Master of Science

Data Science

School of Informatics

University of Edinburgh

2024

Abstract

We present a novel, scalable approach to using multimodal LLMs for creating domain-specific chart derendering datasets. We detail our methodology and apply it to the engineering field to create the first chart derendering dataset representative of the engineering domain. We formalize criteria to assess the quality of chart derendering datasets and use these criteria to analyze the ChartEng dataset. Finally, we lay out plans to scale up the ChartEng dataset in the future and encourage other researchers to implement the proposed methodology to create domain-specific datasets for other disciplines.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Kieran Swenson*)

Acknowledgements

I would like to thank my supervisor Andrew Horne for your guidance and your patience with my ever-changing project directions. I would like to thank my supervisor Vivek Iyer for your willingness to respond and engage with my questions. I would like to thank my secondary faculty supervisor Mirella Lapata for your encouragement and feedback. I would like to thank engineering faculty Sam Tammas-Williams, Antonis Giannopoulos, Tim Drysdale, Anjali Pavar, and Matjaz Vidmar, your enthusiasm helped motivate me and your feedback informed my research directions. I would like to thank my family and friends from home for continually sending their love and support across the Atlantic, your support was invaluable. Finally, a big thank you to all the friends I've made in Edinburgh for making this year so memorable, I'll miss you dearly.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Conducting a systematic review of the related literature is a pre-requisite for beginning novel research in most academic disciplines. Systematic reviews are essential to performing quality research, allowing researchers to survey existing knowledge in the field and identify future directions accordingly [1]. Although essential, the process of conducting a systematic review is notoriously tedious. Researchers must sift through vast quantities of past papers, often numbering in the thousands, and screen each one individually for relevance and valuable insights. As a result, systematic reviews can often take months to complete, delaying new research and consuming valuable funding resources.

The Automated Systematic Review (ASR) project, initiated as a collaboration between the Royal Dick Veterinary School, EDINA, and students from the School of Informatics, aims to alleviate some of the difficulties faced by researchers when conducting systematic reviews. The ASR project streamlines the review process by leveraging machine learning to automate as much of the work as possible. The project has already shown significant promise–in preliminary testing the tool has proven capable of reducing the time taken to conduct reviews in the veterinary field from months to a matter of weeks. Building on this success, the project has expanded its scope with the goal of creating a general purpose tool for automating systematic reviews across disciplines. Currently, the focus of the project is expanding the scope of the tool to the engineering discipline, for which the project is currently receiving funding.

Aside from providing the researcher with valuable perspective on past work, systematic reviews also offer a chance to collect data from relevant papers. This data can then

be aggregated, analyzed, and republished in future work, providing a valuable baseline upon which future research can build. One challenge that arises when attempting to collect data from past work is the prevalence of data presented in graphical format. Graphs, or charts as they are generally referred to in academic literature to distinguish from knowledge graphs, are a fundamental tool for communicating data in academic research. Charts are particularly ubiquitous in engineering, where they are critical for conveying experimental results, as they facilitate the identification complex patterns, trends, and relationships that may be obscured when directly analyzing raw data.

1.2 Problem Statement

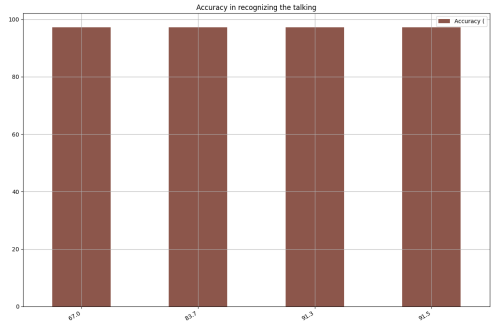
Despite their utility, charts pose a unique challenge to researchers aiming to utilize data from past research. Their visual approach to representing data, while extremely useful for facilitating data analysis, does not allow for easy access to the underlying tabular values. Data presented in charts must first be extracted in tabular format before it can be aggregated or republished. The process of extracting this tabular data is known as chart derendering, although some researchers refer to it as the "Chart-to-Table" task. The process of chart derendering is labor intensive, even with the suite of tools available for the task [2]. Although manual data extraction tools such as WebPlotDigitizer have been shown to produce highly reliable results, this reliability comes at a cost, as extracting data from a single plot typically takes about 15 minutes [3]. Recognizing this challenge, significant effort has been invested into developing AI models to perform automatic chart derendering [4]. Unfortunately, despite significant research into the topic, existing automatic derendering models still lack the accuracy and reliability to be deployed in academic research.

As with many fields of AI, the application of large-scale transformer models to chart derendering led to significant initial performance gains [5, 6]. Characterized by the use of Vision Transformer (ViT) encoders and textual transformer decoders, these models were significantly better suited to chart derendering than past deep learning approaches because of their ability to encode spatial information. This spatial context allowed them to capture relationships between related chart elements such as axes, data points, and legends that are spread throughout the input image. The first models implementing this architecture were actually models trained for more generalized visual language understanding: Pix2Struct and DONUT [7, 8]. Despite no task-specific training, their massive corpuses of visual language data made them well-suited to

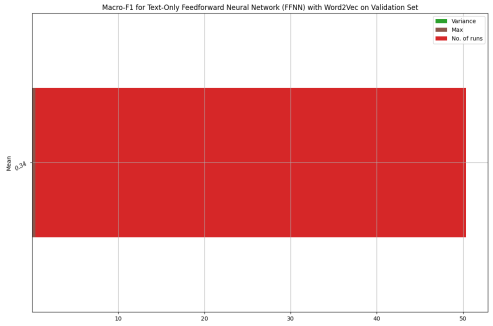
chart derendering, allowing them to considerably outperform more specialized chart derendering models. The initial success of these models inspired multiple fine-tunings on the chart derendering task, in the hopes that task-specific training on chart-table pairs would quickly provide more performance gains [5, 6, 9, 10].

After initial success, model performance quickly plateaued, in large part due to the lack of high-quality datasets of chart-table pairings [11]. Synthetic datasets have been created to alleviate this issue, but fail to emulate the massive variability in chart types, data distributions, and visual styling seen in real world charts [12, 13, 10].

Although the quality of synthetic datasets released has been improving over time, the methodology used to create them has remained somewhat stagnant. Data is either scraped from online sources, or sampled randomly from common statistical distributions. Charts are created using online plotting libraries, primarily matplotlib in Python, and authors explicitly vary parameters controlling input data, chart types, and chart styling through conditional programming [14]. This approach to incorporating variability is severely constrained in terms of scalability and flexibility, as authors must individually identify and define every parameter, all of its permutations, and how that parameter varies in relation to other parameters. Opting to vary parameters in less structured ways is valid as well, but does tend to result in unnatural looking charts, such as the poorly scaled bar charts (Figure 1.1) in the ChartSFT dataset [10].



(a) Simple bar chart taken from ChartSFT dataset. Poorly chosen y-axis limits results in four bars of indistinguishable height.



(b) Bar chart from ChartSFT. The choice of a stacked bar layout results in an unnatural layout and unintellegible data.

Figure 1.1: Bar charts taken from ChartSFT dataset.

Beyond the challenges of introducing variability, another critical issue lies in the development of domain-specific datasets. These domain-specific datasets are crucial because there exists massive variance between the features and styling of charts across domains; charts found in academic research bear little resemblance to those seen in

journalistic articles, for example. A dataset tailored to represent the specific characteristics and visual styling of engineering charts would be a significant step forward on the path to developing a chart derendering model for deployment in the ASR project. Moreover, improvements in the methodology for developing domain-specific datasets would greatly benefit the field as a whole, enabling more effective model training across various specialized areas.

1.3 Project Goal

In this report, we detail the creation of the ChartEng dataset, a dataset of synthetic charts which mirror the features, styling, and data distributions found in charts taken from engineering papers. We propose novel methodology for dataset creation that leverages the potency of the ChatGPT-4o model for multimodal image understanding [15]. Charts taken from real engineering papers are fed into the model and the model is given instructions to reply with Python code to generate a synthetic chart visually similar to the original. This method compels GPT to generate synthetic data that mirrors the distribution of the original chart's data. This synthetic data, when paired with the resulting chart upon execution of the code, results in a chart-table pairing with visual characteristics and data distribution that closely resembles the input chart.

We implement this innovative approach in a scalable pipeline using the GPT-4o API to create the ChartEng dataset. ChartEng contains 14,670 charts representative of the complexities and variability found in engineering research, a few of which are displayed in Figure 1.2 along with the graphs they are meant to emulate. These charts are paired with their underlying tabular data, as well as the code used to generate them. We detail the methodology behind the pipeline, allowing future authors to use our procedure to create domain-specific datasets for other domains. We formalize criteria for evaluating the quality of a chart-table dataset and apply these criteria to the ChartEng dataset. Finally, we acknowledge shortcomings in our approach, notably the robustness of our data extraction methodology, and lay out plans for future work to validate and scale up ChartEng so that it can be deployed to train chart derendering models.

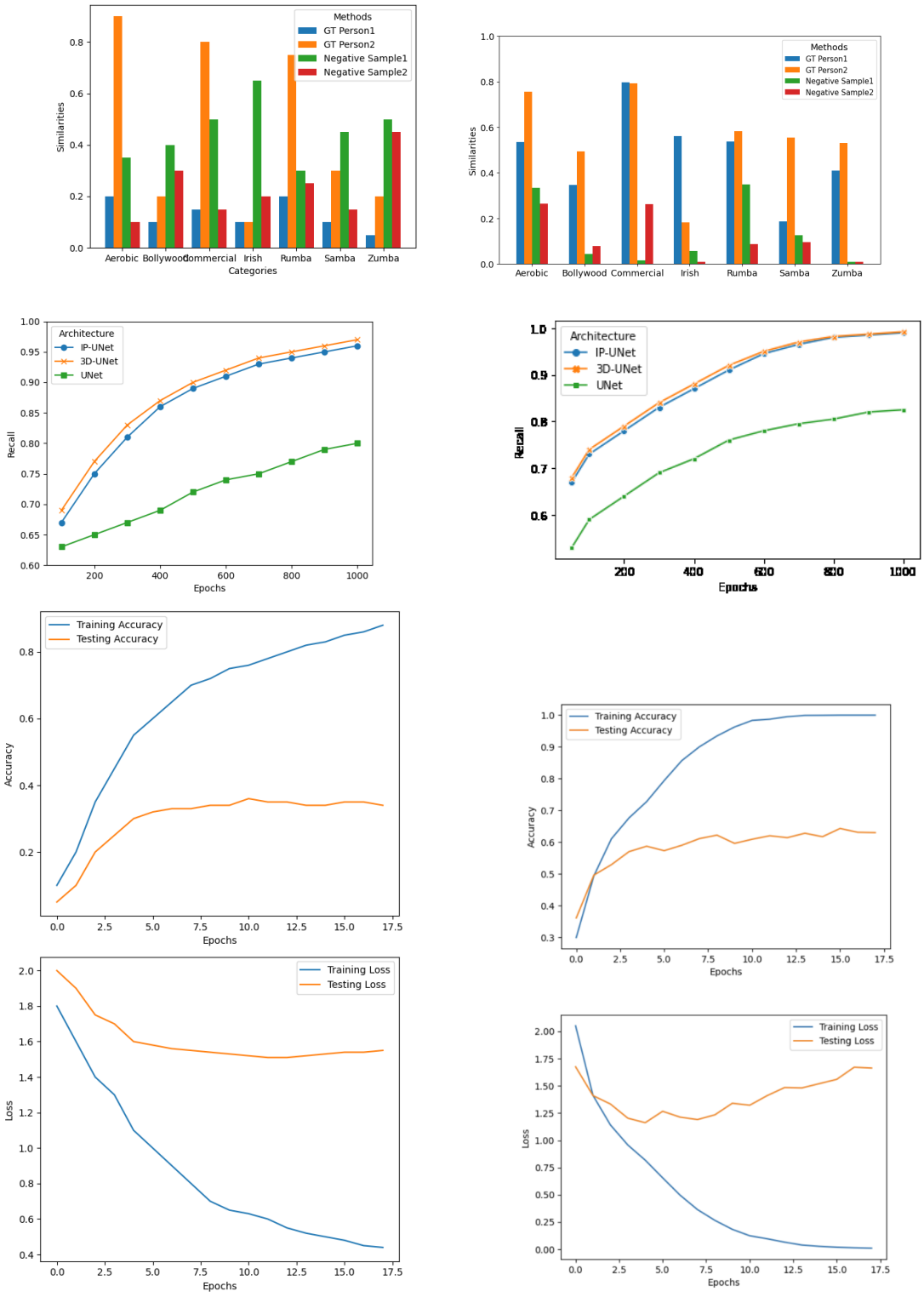


Figure 1.2: Example charts taken from the ChartEng dataset, shown on the left, paired with the original charts input to the ChatGPT-4o model to create them.

Chapter 2

Background

2.1 Manual Chart Derendering

While no longer done using pen, paper, and ruler as it once was, manual chart derendering still requires a substantial degree of human effort. Software tools like PlotDigitizer and WebPlotDigitizer allow the user to upload chart images and extract data with intuitive graphical user interfaces [16, 17]. Users must manually label axes, calibrate the scale of the plot, and select data points individually using the mouse. The paid version of PlotDigitizer offers automatic line tracing and data point detection, but the user must still be involved to select regions of the image to search, label output data, and calibrate the plot scale. The need for user engagement on a plot-by-plot basis makes these tools a significant bottleneck in research applications. Despite their time-consuming nature, the reliability of these data extraction tools continues to make them the preferred choices in academia [3].

2.2 Automatic Chart Derendering

2.2.1 Past Approaches

Early chart derendering approaches relied heavily on heuristic and rule-based methods tailored to specific chart types. One notable example is ChartOCR, which used a pipelined approach combining hand-designed rules, OCR, keypoint detection, and object segmentation to achieve reasonable success across line, bar, and pie charts. Other approaches achieved success by limiting their scope to specific chart types such as bar or line charts [18, 19].

2.2.2 Multimodal Transformer-Based Approaches

Past deep learning approaches struggled with chart derendering because of the inherently local nature of convolutional neural network (CNN) architectures, which struggled to capture the long-range dependencies between chart elements necessary to perform chart derendering [20]. In the past couple of years, research into chart derendering has shifted almost entirely to transformer-based approaches. Unlike CNNs, Vision Transformers (ViTs) encode spatial information by dividing the input image into patches and assigning them labels, before processing them analogously to how words are processed in natural language processing [21, 22]. Through extensive training, the self-attention layers of the ViTs can learn complex spatial dependencies, allowing them to handle tasks that require understanding relationships between various disparate elements within an image.

These powerful new multimodal models do come with a caveat: they require vast amounts of training data. Unfortunately, datasets of chart-table pairs of the quality and scale required to train these models, which contain hundreds of millions or even billions of parameters, do not currently exist. As a result, the first models employing transformer-based architectures to achieve success on chart derendering were actually models trained for more general visual language understanding tasks. The DONUT model, trained for visual document understanding, or parsing document images into text, was the first large-scale multimodal model applied to the chart derendering task [8]. Pix2Struct, a visual language model trained to parse screenshots of webpages into HTML source code, emerged the same year. The effectiveness of these large-scale models, driven by their massive training corpuses—11 million documents for DONUT and 80 million webpage screenshots for Pix2Struct—was evident from the outset [7]. Both models immediately demonstrated their capability in chart derendering, surpassing previous state-of-the-art models on the ChartQA benchmark.

The DONUT model employed the image-encoder-text-decoder architecture which has since become standard across the field. In this architecture, input images are divided into patches of size 16x16 pixels, each of which is encoded into an embedding space by the image encoder. After the encoded patches are given a chance to attend to each other and exchange information, the text decoder then sequentially translates them into output text. Another model employing a similar architecture, Pix2Struct, emerged soon after for the task of parsing screenshots of webpages to HTML source code . Despite not being specifically trained on chart derendering, the power of these large scale models trained on massive training corpuses was evident [7].

Model	Base Model	Size	Chart Derendering Accuracy (RMS score)
Pix2Struct	–	300M	85.9
Donut	–	260M	87.4
Matcha	Pix2Struct	–	89.6
Unichart	DONUT	260M	91.1
ChartLLaMa	LLaMa	13B	90.0
ChartAst-D	DONUT	260M	92.0
ChartAst-S	SPHINX	13B	91.6

Table 2.1: Results of Models on the Chart-to-Table Task (ChartQA) [23]

Since their advent and subsequent open sourcing, multiple research groups have fine-tuned Pix2Struct and DONUT on datasets of chart-table pairs. The first group to successfully fine-tune one of these multimodal foundation models to the task of chart derendering was Liu et. al., who created two separate fine-tunings each with slightly differing training objectives: MatCha and DePlot [6, 5]. The authors further standardized the field of chart derendering by introducing a more robust error metric called Relative Mapping Similarity (RMS). (Following in their footsteps), the Unichart model achieved an impressive 91.1% success on the ChartQA benchmark as a fine-tuning of the DONUT model [9]. The current state-of-the-art is the ChartAst-D model, which employs a DONUT based architecture to achieve 92.0% accuracy on the ChartQA dataset. Other models, such as ChartLlaMa model and ChartAst-S, have been created as fine-tunings of the larger foundation models LlaMA and SPHINX [24], [25], [26]. Even the increased representation power offered by these models did not result in performance gains, however, and current models are still far from the accuracy required for real world applications.

2.3 Chart Derendering Datasets

2.3.1 The ChartQA Dataset

The classic benchmark dataset for evaluating chart derendering models is the ChartQA dataset [27]. Developed by Masry et. al., the ChartQA dataset is prized in the academic community because of its use of real charts and corresponding tabular data. ChartQA is one of few available datasets containing chart-table data pairs taken from (real sources)

Chart Type	Statista	OWID	OECD	Percentage of Dataset
Bar	16,919	507	128	85.04%
Line	2,169	279	103	12.36%
Pie	537	0	0	2.60%
Total	19,625	786	231	100.00%

Table 2.2: Composition of the ChartQA dataset for training chart derendering models, including the percentage of each plot type.

such as Statista, Our World in Data (OWID), and the Organisation for Economic Co-operation and Development (OECD). Table 2.2 details the composition of the dataset while Figure 2.1 shows example plots taken from the dataset.

Perhaps more salient to the academic community, the dataset also contains question-answer pairings for training chart question understanding models capable of performing a variety of tasks such as chart captioning, question answering, summarization and more. This overarching goal of creating AI capable of performing chart understanding is the true motivating force behind research into chart derendering. These LLM-based models implement chart derendering as a first step in a pipeline towards achieving these downstream tasks, and most researchers address the chart derendering task only as a step towards achieving this more lofty goal [6, 24, 9, 10, 28].

This relegation of chart derendering to a mere step on a pipeline seems ill-advised– there is little application for a model summarizing hallucinated data. Regardless of the quality of the captions, summaries, or answers given by these chart understanding AIs, their utility will remain limited until they prove capable of extracting tabular data from input charts with high fidelity.

2.3.2 Issues with Current Benchmarks

The academic community's widespread acceptance of the ChartQA dataset as the universal benchmark for chart derendering belies a number of significant issues. As detailed in Figure 2.2, ChartQA is limited to just bar charts, line charts, and pie charts. The dataset is overwhelmingly composed of bar charts, of which the majority are sourced from the website Statista. This skewed distribution of chart types and sources can bias model training, constraining the ability of models trained and evaluated on the ChartQA dataset to generalize to other unseen chart types and data domains [4].

Figure 2.1: Example charts taken from the ChartQA dataset. Charts show little variation in visual styling. All four include data markers adjacent to data points.

Additionally, the charts in the ChartQA dataset are incredibly uniform when it comes to visual styling. Charts sourced from Statista use the exact same color scheme, fonts, and legend styling, even featuring the same watermarks in the bottom corners. This homogeneity in visual styling makes the dataset a poor metric of a model's ability to perform on charts sourced from other domains such as engineering research papers.

Most concerning for its applicability as a representative benchmark is the fact that the vast majority (about 90%) of charts included in the ChartQA dataset include data markers containing exact data values. Models trained on the ChartQA dataset can learn to read the data values from the labels and associate them with the corresponding points (or bars), which are conveniently located adjacent to the corresponding data points. Given that performance on the ChartQA dataset has plateaued at just above 90%, it is distinctly possible that state-of-the art chart derendering models can primarily attribute their scores on the ChartQA dataset to basic optical character recognition.

2.3.3 Other Datasets

2.3.3.1 PlotQA

The PlotQA dataset is widely used for training chart derendering models and contains 224,000 plots generated from World Bank Open Data and Open Government Data [29]. The dataset lacks diversity in visual styling, with little variation in axes, color schemes, and legends. In plain terms, the charts are just much simpler than charts seen in real world applications.

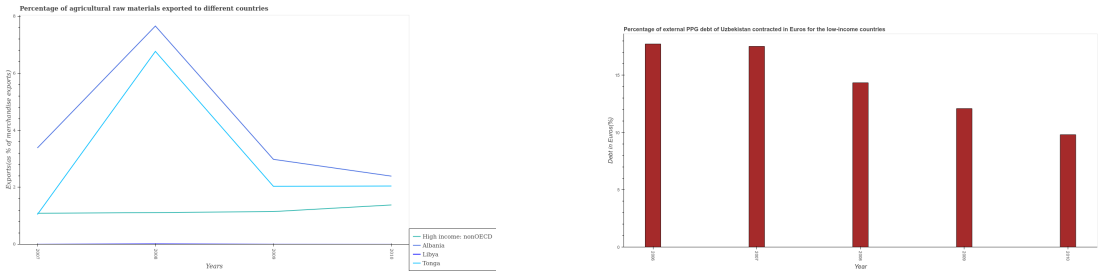


Figure 2.2: Sample charts taken from the PlotQA dataset.

2.3.3.2 ChartSFT

The creators of ChartSFT emphasize the use of real data sourced from academic papers in their synthetic chart creation pipeline. Meng et. al. aggregate data from existing datasets, like ChartQA and PlotQA, and combine it with 132,719 synthetic charts created using tabular data scraped from arXiv research papers to create the largest corpus of chart derendering training data yet assembled [10]. Like the datasets that compose it, ChartSFT suffers from homogeneity in chart types and visual styling. This is not remedied by the addition of the synthetic charts sourced from arXiv, as shown in Figure 2.3. Despite the authenticity of the data, the lack of human oversight in the plotting process leads to unrealistic charts, as evidenced by the plots provided back in the introduction in Figure 1.1.

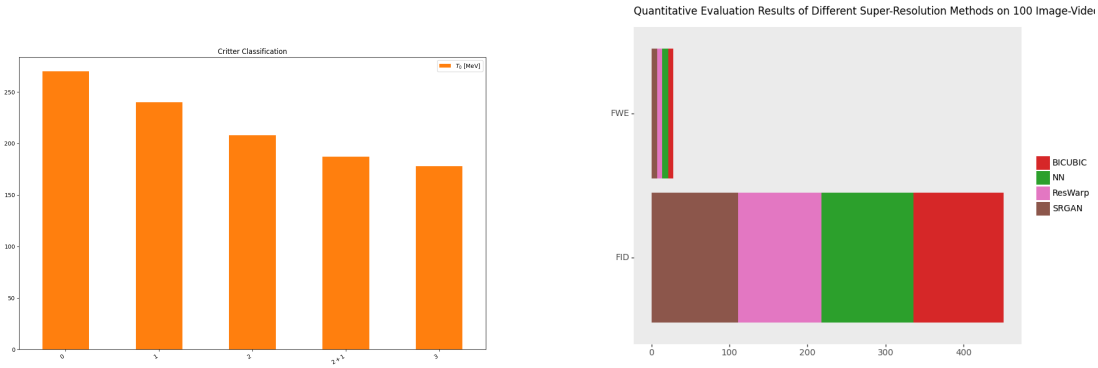


Figure 2.3: Sample bar charts from the arXiv corpus of the ChartSFT dataset. Like ChartQA and PlotQA, the dataset contains primarily bar charts.

2.3.3.3 ChartX

The ChartX dataset is a fully synthetic dataset, meaning the underlying data tables are created rather than taken from real sources. It features a robust synthetic data creation process, incorporating the ChatGPT-4 model [13] to create a dataset spanning 25,000 charts and 18 chart types. Not only does ChartX vary more visual styling parameters than any previous dataset, but it does so in well-thought out ways that do not result in the unnatural charts seen in ChartSFT. Despite these improvements, the dataset still lacks numerous dimensions of variability found in real world domains, such as academic research papers. Sample charts taken from ChartX are included in Figure 2.4.

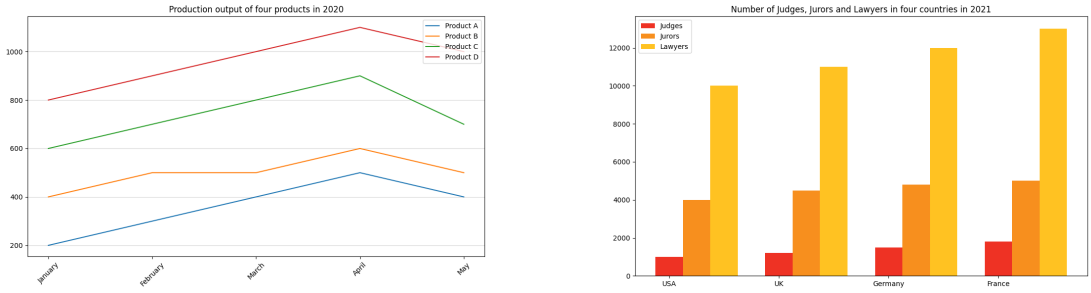


Figure 2.4: Sample charts from the ChartX Dataset. There is significantly more variety in visual styling then in either ChartQA or PlotQA. Like the ChartQA dataset, ChartX does not include scatter plots.

2.3.4 Criteria for a Good Chart Derendering Dataset

In the literature on chart derendering, several points are frequently mentioned as being essential to the quality of a dataset for training chart derendering models. Variation in

chart types and chart visual styling are cited as important to prevent models overfitting to specific charts and visual features [13, 6]. The use of data from real sources is also seen as a major positive, although synthetic data is viewed as an acceptable substitute. Finally, because of the massive data demands of training large transformer models, the size and scalability of a dataset is also of paramount importance. While these points capture some of the relevant criteria for evaluating a dataset, a more proper formalization will be useful for our purposes:

1. **Chart Types**: A robust dataset for training chart derendering models should encompass a variety of chart types. Different chart types encode data in disparate and dissimilar ways, and the inclusion of a variety of chart types in a dataset is vital to ensuring a model will be capable of extracting to data regardless of the format it is encoded in. Furthermore, the distribution of chart types should be balanced to prevent the model from overfitting to chart types that may be over represented in the dataset.

2. **Data Types and Distributions**: A well-constructed dataset should incorporate a mixture of both numerical and categorical data. It should also include variety in the statistical characteristics of the data, such as distribution shape (e.g. normal, uniform, skewed), density of data points, range, and outliers. This variation helps the model generalize across different real-world datasets it may encounter upon deployment.

3. **Chart Visual Styling**: A good dataset should contain meaningful diversity in the aesthetic and structural elements influencing the visual appearance of charts. This variance in visual styling should be apparent both across chart types and within charts of the same type. A high quality dataset of charts should include variance in visual elements, including but not limited to:

    (a) **Data Representation Style (data marker type, style, and coloration)**: Charts can contain enormous variety in the presentation of data. This variety is present across types, as each chart type encodes data in different ways, but also within chart types in the shape, style, and coloration of the data markers. Within line graphs, for example, there is variance in line style (e.g. filled, dashed, dotted), line thickness, marker type (e.g. circles, squares), and color schemes. Data may also be represented in multiple ways within the same plot (e.g. a line plot with a scatter plot overlayed). A diverse

dataset should expose a model to these variations in data representation, training it to interpret and extract data independent of these stylistic choices.

(b) **Representative Scale Information (axes, scale, and tick marks)**: The primary way that charts communicate information about their representative scale is through axes. As such, a chart derendering model must be able to interpret axes in order to extract data points correctly. Including variety in configuration (e.g. standard x-y, categorical axes, multi-axis plots), scale (e.g. linear, logarithmic), and tick mark styling is crucial to ensuring chart derendering models trained on the dataset can robustly identify the scaling of a plot.

(c) **Metadata Information (legends and axis labels)**: In order to ensure that a model can extract metadata effectively across different charts, a dataset must contain variance in the location and style of the chart elements used to communicate metadata. Variation in legend placement (e.g. below, above, overlaying chart) and styling (icon-based legends, color-based legends) ensures that a model is able to correctly label different data series appearing on the same chart. Variation in axis label styling (e.g. font, orientation) ensures robustness in variable naming.

(d) **Additional Graphical Elements**: Charts frequently contain additional graphical elements beyond the basic data representation such as grid lines, error bars, arrows, and annotations. These graphical elements may assist in the extraction of data by aiding the model with scale calibration (e.g. grid lines) or extracting data points (data labels). On the other hand, visual elements may sometimes be unrelated to data extraction and obstruct the model (e.g. annotations, error bars). The inclusion of a variety of types of additional elements ensures that a model learns which visual elements to pay attention to and which to ignore.

4. **Chart Complexity**: The dataset should present a variety of "challenge levels" to chart derendering models. A good dataset should contain a range of chart difficulties, from simple charts to more complex ones. The complexity of a chart can be understood as a function of the complexity of its data distribution (2) and visual styling (3). Data distribution complexity is strongly influenced by the amount of data: more data points, more data series, and more overlapping data points can all increase the complexity of a chart. Complexity in visual styling

is both intuitive and subtle (and requires a more nuanced discussion than we have space for here), so we will proceed assuming the reader can exercise their intuition to determine which elements of visual styling contribute to increasing or decreasing a chart's complexity.

5. **Data (and Metadata) Accuracy and Authenticity**: The dataset should contain exact chart-table pairs. Furthermore, the distribution of data found in the dataset should be representative of the distribution of data in real applications. Metadata, including data labels, category names, plot titles, and data series names, should be variable and representative of typical metadata found in the field. In past literature there is a heavy emphasis on maintaining data and metadata authenticity by including data from real-world sources. Despite this emphasis, it is perfectly reasonable to use synthetic data as long as care is taken to ensure its authenticity.

6. **Size & Scalability**: The larger a chart dataset, the greater its utility as a training set for data-hungry chart derendering models. Datasets of small size may still be useful if their methodology can be scaled [12]. Crucially, the dataset must not compromise on any of the criteria listed above when scaled.

The criteria outlined above provide a more structured framework for evaluating chart derendering datasets. We will evaluate our dataset's performance on the above metrics in the abstract, but also with an eye on mirroring the characteristics seen in the engineering field. For example, for our dataset to be truly representative, it should not just contain variance in chart types and complexity, but should introduce variance of a similar type and at a similar frequency to real charts in engineering papers. We hope that adhering to these criteria in our analysis will ensure we conduct a fair and unbiased evaluation of the ChartEng dataset.

2.4 ArXiv Scraping

No datasets of chart-table pairings in the style of ChartEng have been previously created using the arXiv repository. However, arXiv has been used as the source for constructing various large-scale datasets of scientific figures and associated captions, which have been deployed to train figure and chart understanding models. While none of these datasets contain the chart-table pairings necessary to train chart derendering models, they do demonstrate the potential for using automated classification techniques to filter large volumes of images sourced from arXiv and to identify and extract charts.

The first of such datasets was the SCICAP dataset, a massive dataset containing over two million scientific figures from computer science papers published to the arXiv repository. The authors used a pre-trained CNN-based figure classifier, FigureSeer, to sort the downloaded figures into several categories, including charts, tables, flowcharts, equations, and an "other" category. Unfortunately, the FigureSeer classifier was only capable of performing this sorting with an accuracy of 86%, meaning that the chart category was diluted with non-chart figures [30].

The LineCap dataset, built as a much smaller subset of the SCICAP dataset, contains line charts taken from SCICAP dataset [31]. In order to guarantee the quality of the data, the authors manually inspected the charts to remove incorrectly classified or poor quality figures. Another dataset built from SCICAP, the SciGraphQA dataset included a larger subset of line plots than LineCap, but did not incorporate manual filtering as a validation step [32]. Separately, Li et. al. built the Multimodal Arxiv dataset, a large-scale dataset of figure-caption pairs from arXiv papers to enhance the comprehension capabilities of large vision-language models, but did not classify figures into different types such as charts [33].

Chapter 3

Methodology/Implementation

3.1 Overview

The key steps we undertook to generate the ChartEng dataset are outlined below:

1. **Obtain Images from Engineering Research Papers**: We used the arXiv API to scrape PDFs of research papers from the Electrical Engineering and Systems Science (EESS) category.

2. **Filter Images for Charts**: We implemented the SalesForce BLIP model, a pre-trained image captioning model, to filter out non-chart images [34].

3. **Generate Python Code to Recreate Charts**: We prompted the GPT-4o model to categorize input charts and generate Python code to replicate them.

4. **Process API Responses and Code**: We developed a procedure for retrieving and standardizing GPT outputs. We organized the responses based on the returned graph type categorization.

5. **Chart Generation and Data Extraction**: We plotted the charts in Python and downloaded them as PNG images. We designed a technique for extracting the underlying tabular data from the charts, resulting in tables which accurately reflect the data in their corresponding chart.

3.2 Obtaining Images from Research Papers

3.2.1 Scraping ArXiv

The arXiv repository is the preferred online journal for scraping because of its accessible and well-documented Python API. To ensure domain relevance, we limited our scope to research papers tagged under one of the four subcategories in EESS: Audio and Speech Processing, Image and Video Processing, Signal Processing, and Systems and Control. Although we acknowledge that the EESS category does not emcompass the entire engineering field, we chose to limit our search to it because it is the only subsection of the arXiv repository explicitly containing the term 'Engineering' in the title. One area of future work is to expand the scope of our search to categories beyond these four. Image downloading was performed on a machine with an 8-core CPU and 16 GB of RAM.

3.2.2 Image Processing and Filtering

As a useful pre-filtering step before applying the BLIP model, we chose to include only images with PNG extensions in our dataset. We implemented this step because we noted that the standard output for the plotting libraries most commonly used in engineering (matplotlib, seaborn, matlab) tends to be PNG. Metadata about each image including paper ID and page number, along with metadata about the papers such as publication date and authors, was stored in a JSON file.

Even after filtering out JPEG files, the images downloaded from the API were overwhelming non-chart images, with only an estimated 6.0% of images containing charts. As such, it was essential that we implement an effective filtering process with a low false positive rate. After initial testing, we determined that the false positive rate of pre-trained chart classifiers, such as the FigureSeer classifier (11% false positive rate) implemented in the creation of the SCICAP dataset, was too high for our purposes.

The most effective approach we found was to caption the images using a pre-trained image captioning model and filter the images based on the provided captions. We employed the SalesForce BLIP model, a vision-encoder-text-decoder model trained on millions of image-caption pairs taken from datasets such as COCO and Visual Genome , to generate captions for each scraped image [34, 35, 36]. Captions that contained the keywords "graph" or "plot" were classified as charts and selected for further processing. Further analysis of the performance of the BLIP model is included in Appendix C.

The BLIP model was run with the default max caption length of 20 tokens. The script was run on the DICE (Distributed Informatics Computing Environment) at the University of Edinburgh. We utilized one GTX-2080TI GPU and four CPUs of unknown computing power [37].

3.3 Generating Python Code for Charts

The next step in the pipeline was to generate code to recreate the chart as closely as possible. OpenAI's newest model ChatGPT-4o was chosen for this task due to its state-of-the-art multimodal capabilities [38]. I experimented with free, open source alternatives such as the QWEN-VL-MAX, LLaVA, and InternVL-Chat-V1.5 models, but the charts they generated were overly simplistic and the code often contained errors [39, 40, 41].

3.3.1 API Call Specifics

I used the GPT-4o API chat completions endpoint and the model version updated as of 13/05/2024. API calls were submitted in batches, as OpenAI offers a 50% discount for batch processing. Images were hosted on Google Drive and URL links to the images were included in the API calls to allow the API to access them [42]. API calls were structured as JSONL with each line corresponding to a single input image file, in accordance with OpenAI guidelines. The cost of the API was $2.50 / 1 million input tokens and $7.50 / 1 million output tokens. Images were input at low resolution for a cost of $4.25 / 10,000 input images. API returns were capped at 1000 tokens to ensure overlength responses did not eat into our budget unnecessarily. We used small batches (30 images) to test API usage and prompts before submitting input images in five batches.

3.3.2 Prompting

The final prompt was carefully crafted to maximize the quality of the chart and its similarity to the input chart. We also attempted to minimize the input and output token usage to reduce costs. We found that the model performed best when we listed specific visual features we wanted to be maintained in the output charts. I included a "NONE" response option for the model as a non-response option to further filter out non-chart images that may have been incorrectly categorized by the BLIP model. We prompted

the model to include a comment indicating the chart type, allowing us to organize outputs by chart type. Further discussion of prompt testing, including the full list of prompts tested, is included in Appendix E.

Prompt

System Prompt: You are a highly skilled AI specialized in generating Python code for creating graphs that are visually similar to provided examples. Generate Python code using matplotlib to create a graph resembling the provided image. Ensure the graph type, data distribution, color schemes, line styles, markers, legends, labels, axes, error bars, data markers, etc. match the example. Include a comment at the top with the graph type (e.g., # bar). Do not include any other comments. If the image does not contain transcribable data (e.g., lacks labeled axes, displays only patterns), respond with NONE.

User Prompt: URL link to the image on Google Drive

3.4 Graph Generation

3.4.1 Output Processing

Outputs from GPT-4o were formatted as JSONL files, with each line corresponding to a single input image. Outputs were standardized and extraneous code elements (e.g. ```` ```Python ````) that the model frequently (included) were removed. Folders were created to organize the responses by the graph type categorization given by GPT and responses were converted and saved as Python files. Logging was implemented to record 'NONE' responses. Code files were named with the same name as the input chart images to which they corresponded to.

3.4.2 Chart Plotting

Once the output code files were organized into their respective folders, we executed the code to generate graphs. We began by looping over all subfolders in the code directory and, for each file, modifying the code for graph generation. This involved removing the `plt.show()` call (prompting Python to print the image to terminal) at the end of the code files and replacing it with a `plt.savefig()` call with the appropriate image name

and input and output folder pathways.

3.5 Data Extraction

Data extraction was performed concurrently to chart plotting. We leveraged the object oriented methods provided by matplotlib to interact with the charts and extract data and metadata from them. We designed heuristic methods to handle different chart types and implemented them in a function. To extract data from bar charts, for example, the `extract_plot_data_function` function focuses on the bar containers within the axes, extracting the height of each bar and the bar label. The function also captures relevant information about the visual styling of the chart, such as color scheme, line styling, and marker types, and stores it in a separate metadata file. Because of time constraints, we only validated our approach on the three most common types of charts in the ChartEng dataset: line charts, bar charts, and scatter charts.

We integrated data extraction into the chart plotting script, defining the data extraction function and importing necessary libraries at the top of each script, then adding code to call the function on the chart objects upon execution. Executing the code files generated the charts, called the `extract_plot_data_function` function on the chart to extract data and metadata, before saving the chart and closing the figure. The extracted information was saved in structured formats (CSV for data, JSON for metadata), enabling efficient storage and further analysis.

3.5.1 Summary of the Pipeline

In total, we scraped 296,974 images from 17,127 research papers. After filtering with the BLIP model, we identified 17,904 candidate charts. Of these candidates, 15,857 were identified as charts by GPT. Due to errors in some of the outputted code (and some weird non-responses from the API), this resulted in 14,670 output charts, of which we extracted data from 12,612. Table 3.1 contains a summary of the files present at each stage of the pipeline.

Pipeline Stage	File Type	Number of Files	Percentage of Original Files
Images Scraped	PNG	296,974	–
Candidate Graphs Input to GPT-4o	PNG	17,904	6.0%
Code Files Output by GPT-4o	Python	15,857	5.3%
Charts Generated	PNG	14,670	4.9%
Chart-Table Pairings	CSV	12,612	4.2%

Table 3.1: Number and types of files at different stages of the ChartEng dataset creation pipeline. File numbers are compared to the size of the original corpus of images scraped from arXiv.

Chapter 4

Analysis of the ChartEng Dataset

In this chapter, we analyze the ChartEng dataset based on the criteria we defined in Section 2.3.4. We limit the scope of our analysis to single-chart figures and charts plotted on 2 dimensional axes to facilitate comparison to existing datasets. We begin by examining the distribution of chart types within the dataset, followed by a detailed evaluation of the data distributions and visual styling diversity. We continue on to assess the complexity of the charts, both in terms of data and visual elements, and discuss the accuracy and authenticity of the extracted data. We end by considering the scalability of the ChartEng dataset and comparing it to other chart derendering datasets.

4.1 Chart Types

4.1.1 Distribution of Chart Types

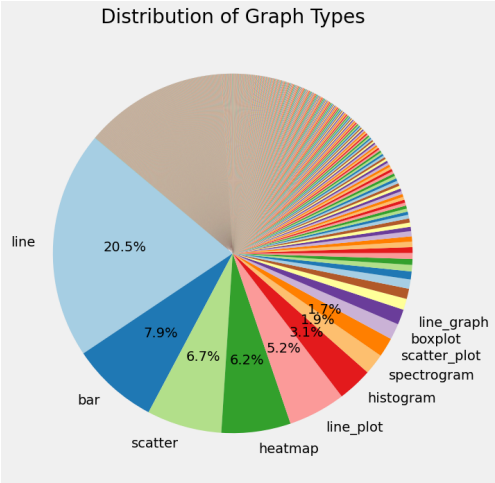
The dataset contains significant variety in graph types, with 2,608 unique categories of charts according to GPT-4o. The model was very faithful to input graph type, rarely outputting graphs of different types to the input graph. Because the model was not given pre-defined categories to sort the graphs into, however, it tended to generate very fine-grained categorizations, creating 2,608 unique graph type categories in total. Visual inspection of the category names and charts confirms that the model segregated)many charts that would be included under one category had the dataset been curated by humans. For example, ‘bar’, ‘bar plot’, ‘bar with error’, and ‘bar with error bars’ are all separate categories in ChartEng. The largest categories, measured by number of output charts, are ‘line’ (3,090), bar’ (1165), ‘scatter’ (984), ‘heatmap’ (880), ‘line plot’ (783), and ‘histogram’ (485). The distribution is highly skewed with a long tail–

2,213 categories have five or fewer graphs and 1,870 contain just one graph. Figure 4.1a breaks down the composition of the dataset by graph type.

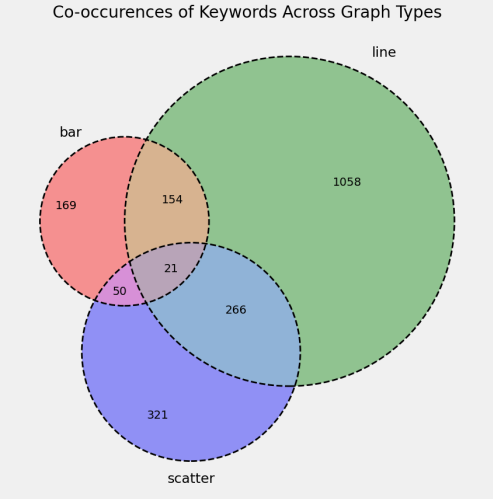
4.1.2 Overlap in Chart Type Categorizations

While the variety of graph types in the ChartEng dataset is an advantage, it is not ideal to have graphs of the same type spread across so many different categories. Ideally, we would like to group categories containing graphs of the same overarching type together. One potential approach is to perform keyword-based grouping of the categories. This is difficult, however, as the categories generated by the model overlap significantly with each other. Figure 4.1b details the overlap between the three most common keywords (aside from 'graph' or 'plot') appearing in category names in the ChartEng dataset.

For example, choosing to group all categories containing the term 'line' under the larger umbrella of line graphs would result in the inclusion of 154 categories which also include the term 'bar,' 266 categories with 'scatter,' and 21 categories that have all three terms. Deciding how to build higher level groupings of these fine-grained categories is one area where future work can help improve the ChartEng dataset.



(a) Distribution of graph types in ChartEng, ordered by number of output graphs. Smaller categories are left unlabelled.



(b) Venn diagram displaying overlap between graph type categories containing the terms 'line,' 'scatter,' and 'bar.'

Figure 4.1

4.2 Data Types and Distributions

While the data in the ChartEng dataset is all synthetic, the data types and distributions are meant to mirror those found in real engineering research. As with graph type, in our analysis the model tended to be faithful to the data types of the input graphs. As such, we believe ChartEng contains numerical and categorical data in the same proportions as the domains of engineering surveyed.

The more difficult element for the model to replicate was the input data distribution. Of course, in order to exactly replicate the input data distribution, the model would need to be capable of derendering the chart. Given the difficulty of chart derendering, it should be unsurprising that the model was incapable of faithfully recreating input data distributions. This inability to replicate the data distribution is often not particularly significant. Figures 4.2 and 4.3 contain example charts where, despite significant differences between the data distribution of the input and output charts, these differences do not affect other important features of the chart such as its visual styling or complexity.

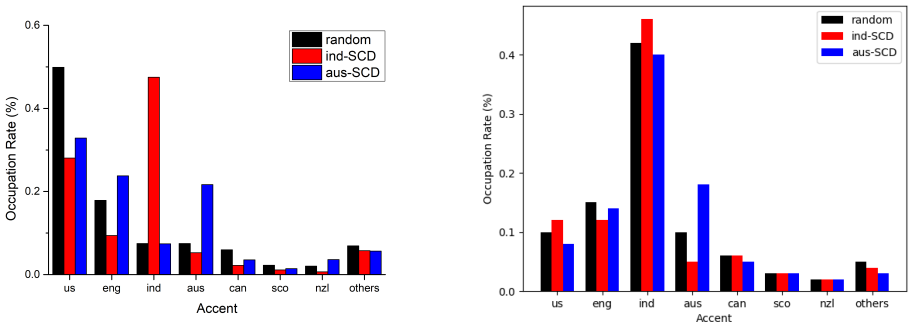


Figure 4.2: Comparison of original (left) and GPT generated (right) bar charts occupation rate for speakers with different accents. Although the model maintained the chart type, data type, and visual styling, the data distribution is shifted from a left-skewed distribution to a more centrally-peaked distribution.

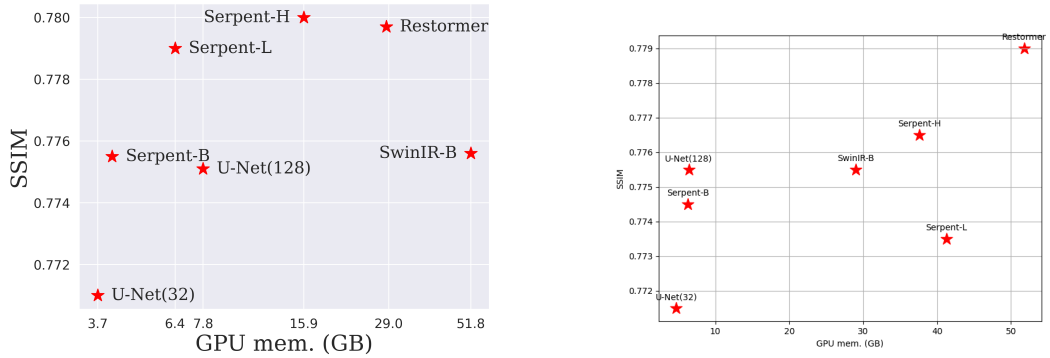


Figure 4.3: Comparison of original (left) and generated (right) scatter plots displaying GPU memory versus SSIM. The model maintains chart type, data type, and some aspects of visual styling, such as marker type and marker color, and data labels. Individual data points may be (rearranged) significantly, but the distribution of the points is similar.

4.3 Chart Visual Styling

The ChartEng dataset contains the largest variation in visual styling of any open source chart derendering dataset. It is the only available dataset that includes significant variability across all four subcategories of visual styling delineated in Section 2.3.4: data representation style, representative scale information, metadata information, and additional graphical elements. Figures 4.4-4.7 contain example charts from the ChartEng dataset, highlighting its extensive variability in visual styling.

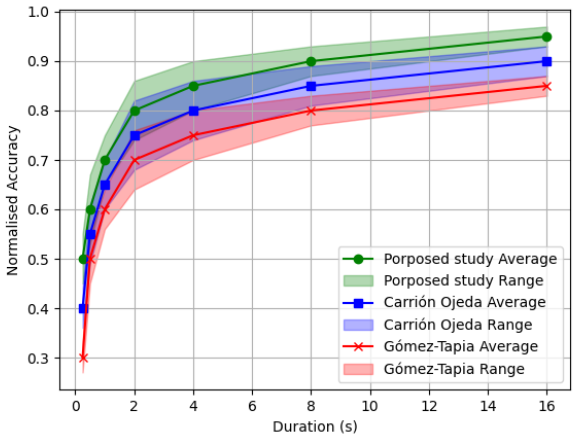


Figure 4.4: Line chart from ChartEng plotted on linear x-y axes. Data series differentiated by color and marker style. Metadata information encoded in icon-based legend overlayed on plot and axis labels. Error regions and gridlines included as additional graphical elements.

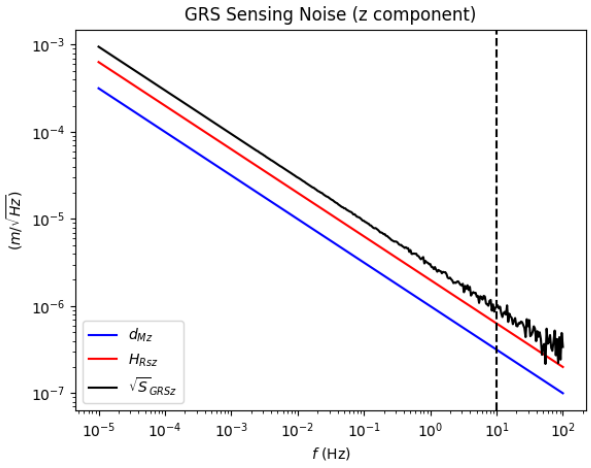


Figure 4.5: Line chart plotted on log-log x-y axes. Similar data differentiation and metadata encoding, but variation in font choice. Vertical dashed line indicating threshold. The inclusion of log scale axes in the ChartEng dataset was a point of emphasis among engineering faculty.

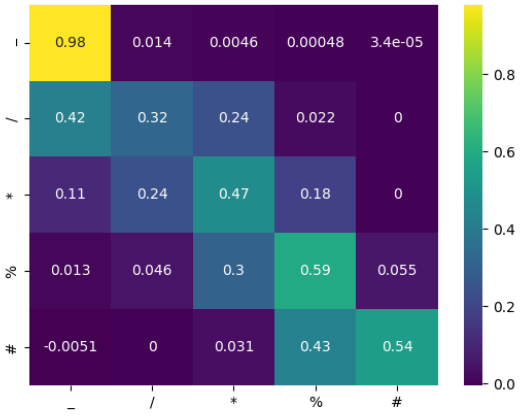


Figure 4.6: Heatmap plotted on labeled, categorical axes. Data encoded using color gradients, but also delineated via data markers. Continuous color gradient legend included to the right of chart. Data markers interchange between decimal and scientific notation.

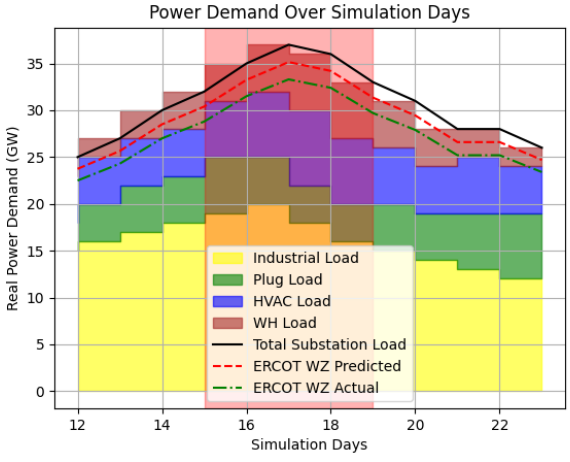


Figure 4.7: Area chart, another problematic chart type mentioned by engineering faculty, with line chart overlayed. Data encoded using lines and color-coded areas, differentiated by color and line style. Metadata communicated via icon-based legend.

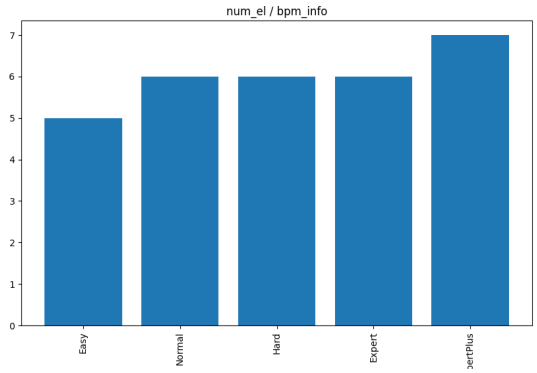


Figure 4.8: Low data complexity and visual complexity bar chart. Single variable plotted on the traditional bar chart category-height x-y axes. No metadata information encoded in color scheme, axis labels, or legend. No additional graphical elements.

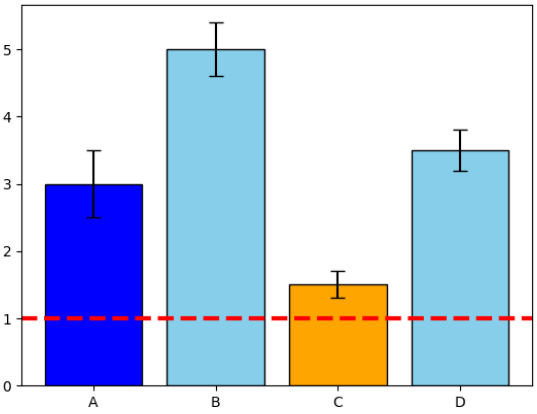


Figure 4.9: Low data complexity chart with higher visual complexity. Introduces variance in bar color as well as additional graphical elements such as error bars and a red baseline. Still single variable with no legend information.

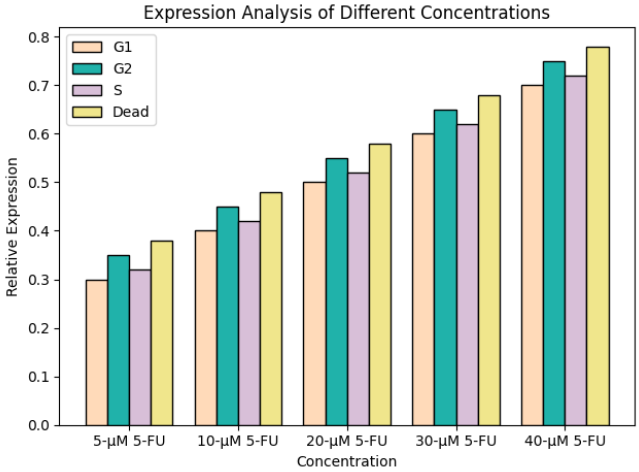


Figure 4.10: Higher data complexity bar chart. Includes multiple bars for each category. Legend overlayed on chart. Metadata information encoded via variation in bar color.

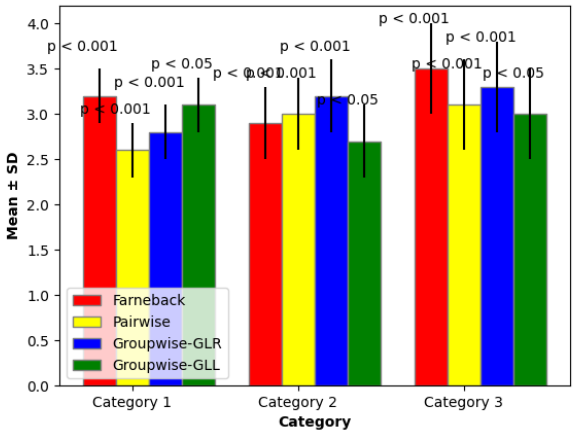


Figure 4.11: High data and visual complexity bar chart. Multiple bars per category distinguished by color. Color-based legend overlayed directly on top of bars. Includes error bars and annotations partially obscuring the top of data bars.

4.4 Complexity

4.4.1 Data Complexity

4.4.1.1 Shifing Data Distributions

The ChartEng dataset contains significant variance in chart complexity within chart types. We can divide our analysis of complexity in terms of data complexity and visual complexity. As discussed in Section 4.3, GPT-4o struggled to replicate the data distributions seen in input charts. Figures 4.2-4.3 contain examples of input-output chart pairings where GPT struggled to maintain the distribution of data points without meaningful consequences.

4.4.1.2 High Complexity Charts

Unlike Figures 4.2-4.3, Figures 4.12 and 4.13 contain an example where the distribution of the output chart shifts enough that it lowers the chart's complexity. Whether because of an inability to interpret the high complexity of the data in the input chart, an inability to write code to mirror it, or some combination of the two, the model fails to replicate the data distribution in the output chart. This situation where the model simplified the distribution of a complex input chart arose frequently enough that we concluded that the ChartEng dataset, on the average, contains charts with lower complexity than the input charts.

4.4.1.3 Implications

Although the shift in the overall distribution of chart complexities seen in ChartEng may reduce its fidelity to the input distribution, there is an argument to be made that this shift may actually improve the quality of the dataset for training chart derendering models. This is because there is a meaningful distinction between charts containing exact data values that can be derendered and charts whose underlying data is so obscured, such as the input charts seen in Figure 4.12 and Figure 4.13 , that they only communicate visual patterns. Charts from the latter category, while they certainly belong in a representative dataset of charts from engineering papers, do not belong in a dataset meant to be representative of the charts to which researchers might reasonably apply a chart derendering model. As the goal of the ChartEng dataset is to facilitate the creation of chart derendering models, we argue that simplifying the data distributions of input

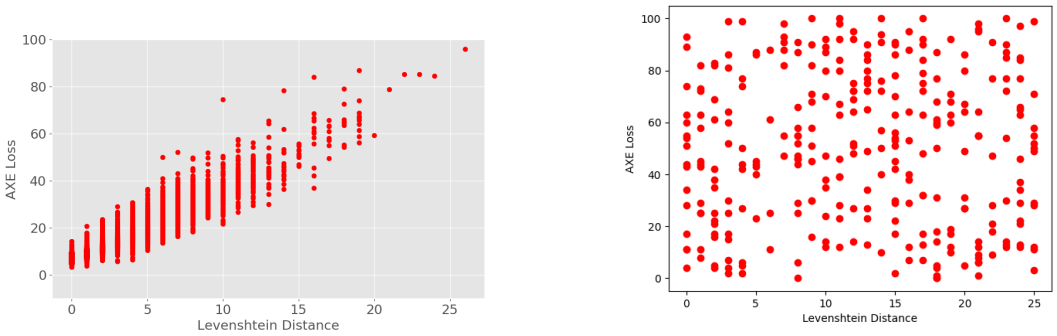


Figure 4.12: Comparison of original (left) and generated (right) charts displaying a shift in data distribution from a more complex distribution containing more overlapping data to a less complex distribution with more separation between points. The chart on the right may actually make for a more useful training sample because it contains points that can be distinguished from one another.

charts can actually be a positive byproduct of the shortcomings of GPT-4o's multimodal understanding.

4.4.2 Visual Styling Complexity

On the whole, GPT succeeded at replicating the majority of visual styling elements in the output charts. Like data points, these elements were often shifted around throughout the output chart, which did not generally affect the complexity of the charts, nor did it bias the distribution of chart styling complexity within ChartEng as a whole. A couple types of input variability that did cause model the model to struggle are listed below:

1. Elements that were overlapping or obscured (data series, annotations, data markers, etc.)

2. Textual variation, such as font and type size. GPT used matplotlib's default font and type size for most charts, even when the resulting text did not resemble the input chart.

3. Variation in line thickness and marker thickness (although notably it did not struggle with color scheme or marker style).

These sources of visual styling differences between the input chart and output charts, among others, may reduce the overall visual styling complexity somewhat. To be somewhat blunt, if concerns around chart derendering models trained on the ChartEng

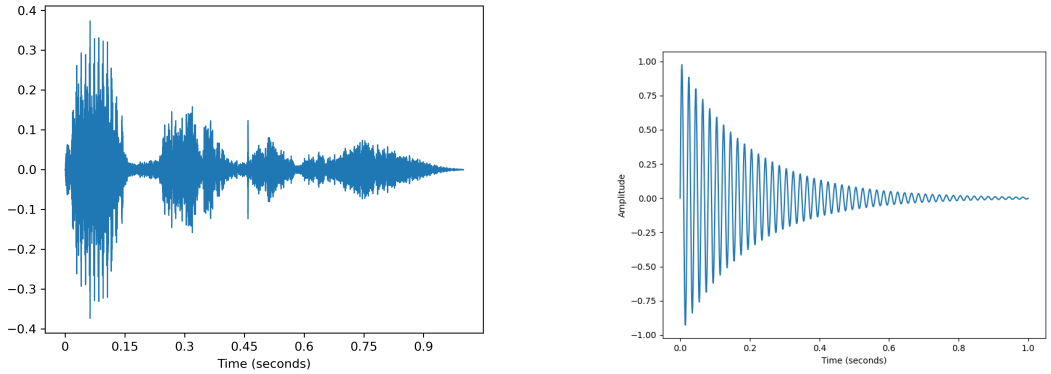


Figure 4.13: Comparison of original (left) and generated (right) line charts displaying a frequency signal over time. Although the model fails to emulate the rapid, irregular fluctuations seen in the original chart, the output chart it creates is still a useful chart to include in the ChartEng dataset.

dataset are about their robustness to charts containing bold text, we will know we have succeeded.

4.5 Data Accuracy and Authenticity

4.5.1 Accuracy

Because of my approach of asking GPT-4o to output code and then replotting that code on my local machine, we can guarantee an exact match between code and generated charts. This may indeed prove useful for training models in the vein of the original Pix2Struct model, which derenders screenshots into HTML code, capable of parsing input charts into Python code.

We did generate chart-table pairings as well, using the methodology discussed in Section 3.5. Currently, however, due to the lack of more comprehensive validation testing, we cannot guarantee the accuracy of our chart-table pairings. This lack of data accuracy is one significant drawback of the ChartEng dataset currently, but further improvements to the data extraction methodology and more extensive validation testing should clear up these remaining concerns.

4.5.2 Authenticity

The data in the ChartEng dataset, though technically synthetic, has been specifically crafted to mirror data plotted in authentic engineering charts. Thus, while the data

does not originate from actual experiments or studies, it emulates the data types and distributions typical of engineering research. While it is not completely faithful to the data distributions of the original graphs, as discussed in Section 4.2, this mirroring process allows the synthetic data to approximate the authenticity of real-world data, up to the limits of current multimodal LLM technology. As such, we believe that the ChartEng dataset can be treated as functionally representative of the domain of engineering charts.

In Figures 4.2 and 4.3, for example, both input and output charts contain valid data distributions that could exist in engineering data. Neither the input chart or the output chart is obviously more representative of the engineering domain, at least not in a way that we believe will affect the performance of chart derendering models trained on the ChartEng dataset.

4.6 Size & Scalability

With just 14,670 charts, ChartEng is smaller than most existing chart derendering datasets. However, our reproducible methodology coupled with the vast resources available on arXiv means that ChartEng is highly scalable. The pipeline is computationally efficient and requires modest computational resources. The only remaining roadblock to scaling the pipeline is the cost of the use of the proprietary GPT-4o model. These costs could be eliminated by exploring open-source alternatives in the future, as discussed in Section 5.2. As soon as improvements in these open-source models allow them to perform adequately as substitutes for GPT-4o, we plan on scaling up the ChartEng dataset to the size necessary for fine-tuning chart derendering models. We also plan on open-sourcing our code, allowing others to implement our methodology to create large-scale, domain-specific datasets across academic fields and push forward the field of chart derendering more broadly. Table 4.1 contains a breakdown of how the composition of ChartEng compares to other common chart derendering datasets.

Dataset	Number of Charts	Domain	Real Charts?	Real Data?	Scalable?
ChartQA	20,642	Politics	Yes	Yes	No
PlotQA	224,377	–	No	Yes	Yes
ChartX	48,000	–	No	No	Yes
ChartSFT	132,719	Academia	No	Yes	Yes
ChartEng (ours)	14,670	Engineering	No	No	Yes

Table 4.1: Comparison of ChartEng to other state-of-the-art chart derendering datasets.

Chapter 5

Future Work

5.1 Validating and Improving Data Extraction

We did not implement a formal framework for validating the chart-table pairs in the ChartEng dataset. Further validation testing of the tables extracted will be necessary before the dataset can be deployed to train or test chart derendering models. Building frameworks for ensuring the robustness of the output chart-table pairs will also be crucial before ChartEng dataset or other domain specific datasets created using our methodology are deployed in real applications.

We would also recommend that future work begin by improving the robustness of the data extraction methodology. Expanding the logic of the data extraction method will allow for extracting data from more complex chart types, increasing the scale and utility of the ChartEng dataset.

5.2 Open Source Models for Chart Replication

Given the cost of using proprietary models like GPT-4o, exploring open source alternatives will be required before truly large-scale datasets can be created using our methodology. Open source models have made incredible strides over the past year, closing the gap on the industry leading ChatGPT models. One exciting alternative is the InternVL2-Llama3-76B model, which recently surpassed the GPT-4o model on the OpenCompass leaderboard for multimodal LLM models [43, 44]. While the authors are careful to include a disclaimer that "this score only captures part of a model's performance" and recognize that "there is still a significant gap between our model and GPT-4o in areas such as instruction following, user experience, pure text processing

capabilities, and overall comprehensiveness," the fact that an open source model is competitive with proprietary models is nonetheless exciting for future research avenues. Testing the performance of the InternVL2-Llama3-76B model and other future open source models on the chart replication task will allow researchers to determine whether our methodology can be scaled without incurring significant costs or sacrificing output chart quality.

5.3 Creation of Other Domain-Specific Datasets

The methodology developed for generating the ChartEng dataset can be easily adapted to other domains. Fields such as biomedical research, environmental sciences, and economics frequently employ charts that differ in style and complexity from those in engineering. Applying our methodology to charts taken from research in these fields will allow researchers to create domain-specific datasets that enhance the accuracy and reliability of chart derendering models in various scientific areas.

Expanding the scraping process to include papers from repositories such as PubMed for medical research or RePEc for economics would facilitate the creation of datasets that accurately reflect the unique characteristics of each field [45, 46]. This approach broadens the scope of chart derendering models and improves their performance by exposing them to the diverse chart types and data distributions specific to each scientific domain.

5.4 Testing and Training Chart Derendering Models

The ChartEng dataset provides a valuable resource for the evaluation and fine-tuning of existing chart derendering models. Conducting baseline tests with state-of-the-art chart derendering models on ChartEng should provide a more accurate assessment of their capacity to manage the complexity and variability characteristic of engineering charts. Should these models struggle, fine-tuning thems on ChartEng should yield improvements in accuracy, particularly in the extraction of data from chart types contained in ChartEng that are not present in their training sets.

Chapter 6

Conclusion

In its current form, the ChartEng dataset does not solve the lack of a domain-specific dataset for the field of engineering. However, the primary contribution of this work lies not in the final dataset but in the introduction of a novel methodology for dataset creation. We believe that our approach of using a multimodal language model to generate synthetic data that closely mirrors real-world engineering data has potential to push the broader chart derendering field forward. Although the scalability of our method is limited by the use of the the GPT-4o model, we expect that improvements in multimodal models will soon ameliorate the need for proprietary technology.

We are particularly excited about the potential for this approach to be applied across domains to create chart derendering datasets. The strength in our approach lies in its flexibility; our pipeline can be implemented without modification to create representative datasets tailored to any field of study. To support further research and development, we have open-sourced both the code and the ChartEng dataset on GitHub. We hope that this will support other researchers in the creation of new datasets, advancing the field of chart derendering as a whole.

Bibliography

[1] Kieran Swenson. Automated systematic review: Data extraction from graphs in research papers. Technical report, Informatics Project Proposal, 2024. Supervised by Prof Andrew Horne, IPP Tutor: Dr Douglas Armstrong.

[2] Ali Mazraeh Farahani, Peyman Adibi, Mohammad Saeed Ehsani, Hans-Peter Hutter, and Alireza Darvishy. Automatic Chart Understanding: A Review. *IEEE Access*, 11:76202–76221, 2023.

[3] Daniel Drevon, Sophie R. Fursa, and Allura L. Malcolm. Intercoder Reliability and Validity of WebPlotDigitizer in Extracting Graphed Data. *Behavior Modification*, 41(2):323–339, March 2017. Publisher: SAGE Publications Inc.

[4] Kung-Hsiang Huang, Hou Pong Chan, Yi R. Fung, Haoyi Qiu, Mingyang Zhou, Shafiq Joty, Shih-Fu Chang, and Heng Ji. From pixels to insights: A survey on automatic chart understanding in the era of large foundation models, 2024.

[5] Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Martin Eisenschlos. MatCha: Enhancing Visual Language Pretraining with Math Reasoning and Chart Derendering, May 2023. arXiv:2212.09662 [cs].

[6] Fangyu Liu, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. DePlot: One-shot visual language reasoning by plot-to-table translation, May 2023. arXiv:2212.10505 [cs].

[7] Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2Struct: Screenshot Parsing as Pretraining for Visual Language Understanding. In *Proceedings of the 40th International Conference on Machine Learning*, pages 18893–18912. PMLR, July 2023. ISSN: 2640-3498.

[8] Geewook Kim, Teakgyu Hong, Moonbin Yim, Jeongyeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. Ocr-free document understanding transformer, 2022.

[9] Ahmed Masry, Parsa Kavehzadeh, Xuan Long Do, Enamul Hoque, and Shafiq Joty. UniChart: A Universal Vision-language Pretrained Model for Chart Comprehension and Reasoning, October 2023. arXiv:2305.14761 [cs].

[10] Fanqing Meng, Wenqi Shao, Quanfeng Lu, Peng Gao, Kaipeng Zhang, Yu Qiao, and Ping Luo. ChartAssisstant: A Universal Chart Multimodal Language Model via Chart-to-Table Pre-training and Multitask Instruction Tuning, February 2024. arXiv:2401.02384 [cs].

[11] Rabah A Al-Zaidy, Sagnik Ray Choudhury, and C Lee Giles. Automatic Summary Generation for Scientific Data Charts.

[12] Brendan Artley. GenPlot: Increasing the Scale and Diversity of Chart Derendering Data, June 2023. arXiv:2306.11699 [cs].

[13] Renqiu Xia, Bo Zhang, Hancheng Ye, Xiangchao Yan, Qi Liu, Hongbin Zhou, Zijun Chen, Min Dou, Botian Shi, Junchi Yan, and Yu Qiao. Chartx & chartvlm: A versatile benchmark and foundation model for complicated chart reasoning, 2024.

[14] J. D. Hunter. Matplotlib: A 2d graphics environment, 2007.

[15] OpenAI. Chatgpt 4.0: Gpt-4 api, 2024. Accessed: 2024-08-17.

[16] PlotDigitizer. Plot Digitizer.

[17] Ankit Rohatgi. Webplotdigitizer: Version 3.0. *https://automeris.io/WebPlotDigitizer*, 2011. Accessed: 2024-08-17.

[18] Hajime Kato, Mitsuru Nakazawa, Hsuan-Kung Yang, Mark Chen, and Bjorn Stenger. Parsing Line Chart Images Using Linear Programming. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2553–2562, Waikoloa, HI, USA, January 2022. IEEE.

[19] Chinmayee Rane, Seshasayee Mahadevan Subramanya, Devi Sandeep Endluri, Jian Wu, and C. Lee Giles. ChartReader: Automatic Parsing of Bar-Plots. In *2021*

IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI), pages 318–325, Las Vegas, NV, USA, August 2021. IEEE.

[20] Shufan Li, Congxi Lu, Linkai Li, and Haoshuai Zhou. Chart-RCNN: Efficient Line Chart Data Extraction from Camera Images, November 2022. arXiv:2211.14362 [cs].

[21] 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, 2021.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[23] Fanqing Meng, Wenqi Shao, Quanfeng Lu, Peng Gao, Kaipeng Zhang, Yu Qiao, and Ping Luo. Chartassisstant: A universal chart multimodal language model via chart-to-table pre-training and multitask instruction tuning, 2024.

[24] Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. Chartllama: A multimodal llm for chart understanding and generation, 2023.

[25] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[26] Yuqing Ye et al. Sphinx-x: Scaling large multimodal models through optimized training pipelines. In *International Conference on Learning Representations (ICLR)*, 2023.

[27] Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A Benchmark for Question Answering about Charts with Visual and Logical Reasoning, March 2022. arXiv:2203.10244 [cs].

[28] Zhi-Qi Cheng, Qi Dai, and Alexander G. Hauptmann. ChartReader: A Unified Framework for Chart Derendering and Comprehension without Heuristic Rules.

In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 22145–22156, Paris, France, October 2023. IEEE.

[29] Nitesh Methani, Pritha Ganguly, Mitesh M. Khapra, and Pratyush Kumar. PlotQA: Reasoning over Scientific Plots, February 2020. arXiv:1909.00997 [cs].

[30] Noah Siegel, Zachary Horvitz, Roie Levin, Santosh Divvala, and Ali Farhadi. FigureSeer: Parsing Result-Figures in Research Papers. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, volume 9911, pages 664–680. Springer International Publishing, Cham, 2016. Series Title: Lecture Notes in Computer Science.

[31] Anita Mahinpei, Zona Kostic, and Chris Tanner. Linecap: Line charts for data visualization captioning models, 2022.

[32] Shengzhi Li and Nima Tajbakhsh. Scigraphqa: A large-scale synthetic multi-turn question-answering dataset for scientific graphs, 2023.

[33] Lei Li, Yuqi Wang, Runxin Xu, Peiyi Wang, Xiachong Feng, Lingpeng Kong, and Qi Liu. Multimodal arxiv: A dataset for improving scientific comprehension of large vision-language models, 2024.

[34] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation, February 2022. arXiv:2201.12086 [cs].

[35] Silvia Rostianingsih, Alexander Setiawan, and Christopher Imantaka Halim. COCO (Creating Common Object in Context) Dataset for Chemistry Apparatus. *Procedia Computer Science*, 171:2445–2452, January 2020.

[36] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations, February 2016. arXiv:1602.07332 [cs].

[37] University of Edinburgh. Dice (distributed informatics computing environment), 2024. Accessed: 2024-08-17.

[38] Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, Dahua Lin, and Kai Chen. Vlmevalkit: An open-source toolkit for evaluating large multi-modality models, 2024.

[39] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond, 2023.

[40] Haotian Liu, Chen Yeh, Angela Zeng, et al. Llava: Large language and vision assistant, 2023. Accessed: 2024-08-17.

[41] Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, Ji Ma, Jiaqi Wang, Xiaoyi Dong, Hang Yan, Hewei Guo, Conghui He, Botian Shi, Zhenjiang Jin, Chao Xu, Bin Wang, Xingjian Wei, Wei Li, Wenjian Zhang, Bo Zhang, Pinlong Cai, Licheng Wen, Xiangchao Yan, Min Dou, Lewei Lu, Xizhou Zhu, Tong Lu, Dahua Lin, Yu Qiao, Jifeng Dai, and Wenhai Wang. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites, 2024.

[42] Google. Google drive api, 2024. Accessed: 2024-08-17.

[43] OpenCompass Contributors. Opencompass: A comprehensive benchmark and evaluation platform for multimodal large language models. https://github.com/OpenCompass, 2023. Accessed: 2024-08-23.

[44] InternVLM Team. Internvlm2-llava: Scaling vision-language models with optimized training and large data. https://internvlm.org/, 2024. Accessed: 2024-08-23.

[45] U.S. National Library of Medicine. Pubmed overview. https://pubmed.ncbi.nlm.nih.gov/, 2024. Accessed: 2024-08-23.

[46] RePEc. Research papers in economics (repec) overview. https://repec.org/, 2024. Accessed: 2024-08-23.

[47] korazer/chart-test-classify · Hugging Face.

[48] Anurag Dhote, Mohammed Javed, and David S Doermann. A survey and approach to chart classification, 2023.

[49] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[50] Lang Cao. Learn to refuse: Making large language models more controllable and reliable through knowledge scope limitation and refusal mechanism, 2024.

[51] Genglin Liu, Xingyao Wang, Lifan Yuan, Yangyi Chen, and Hao Peng. Examining llms' uncertainty expression towards questions outside parametric knowledge, 2024.

Appendix A

Chart Derendering Datasets

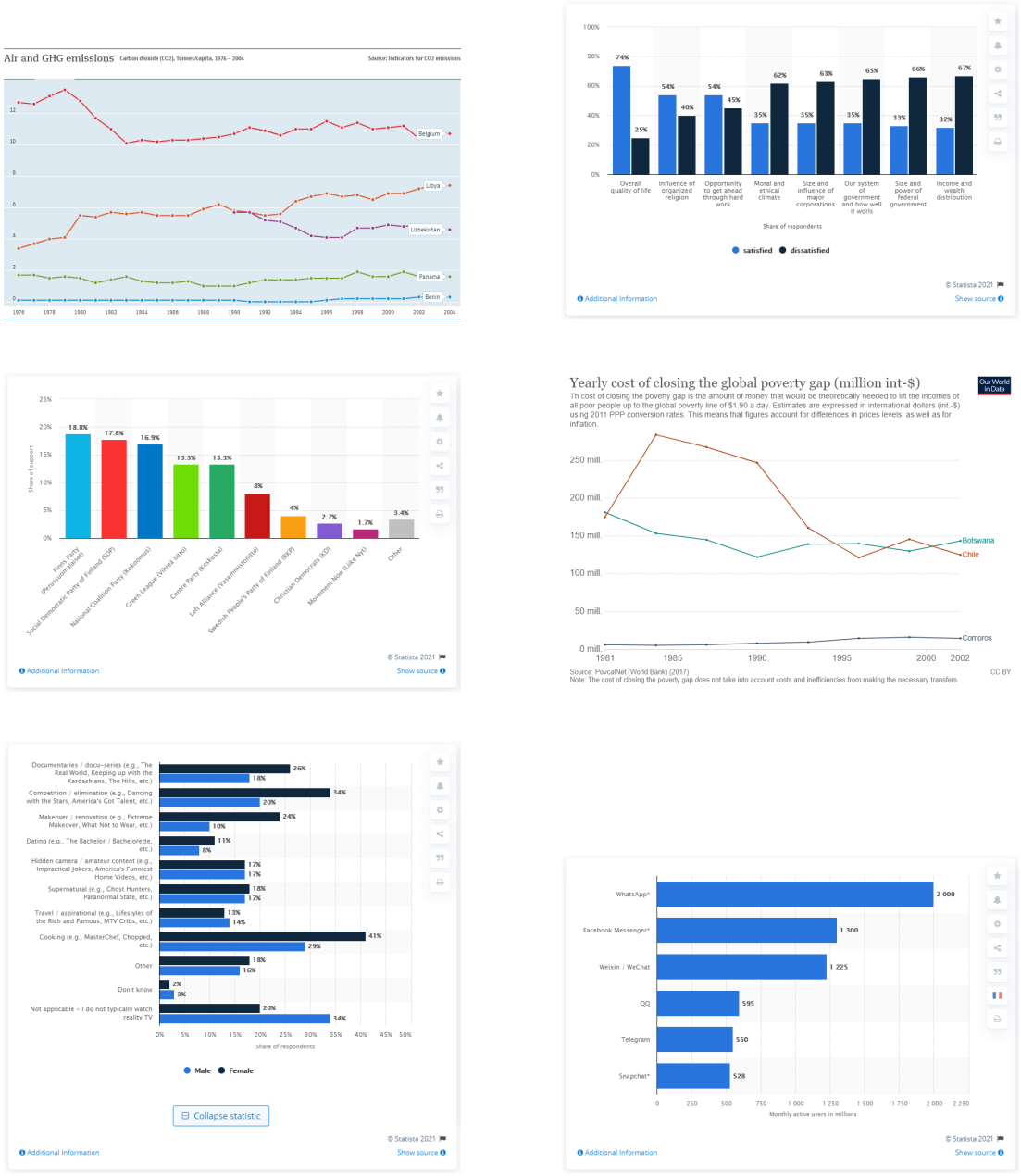


Figure A.1: Additional examples of charts taken from the ChartQA Dataset displaying limited variability in visual styling and prevalence of exact numerical data labels.

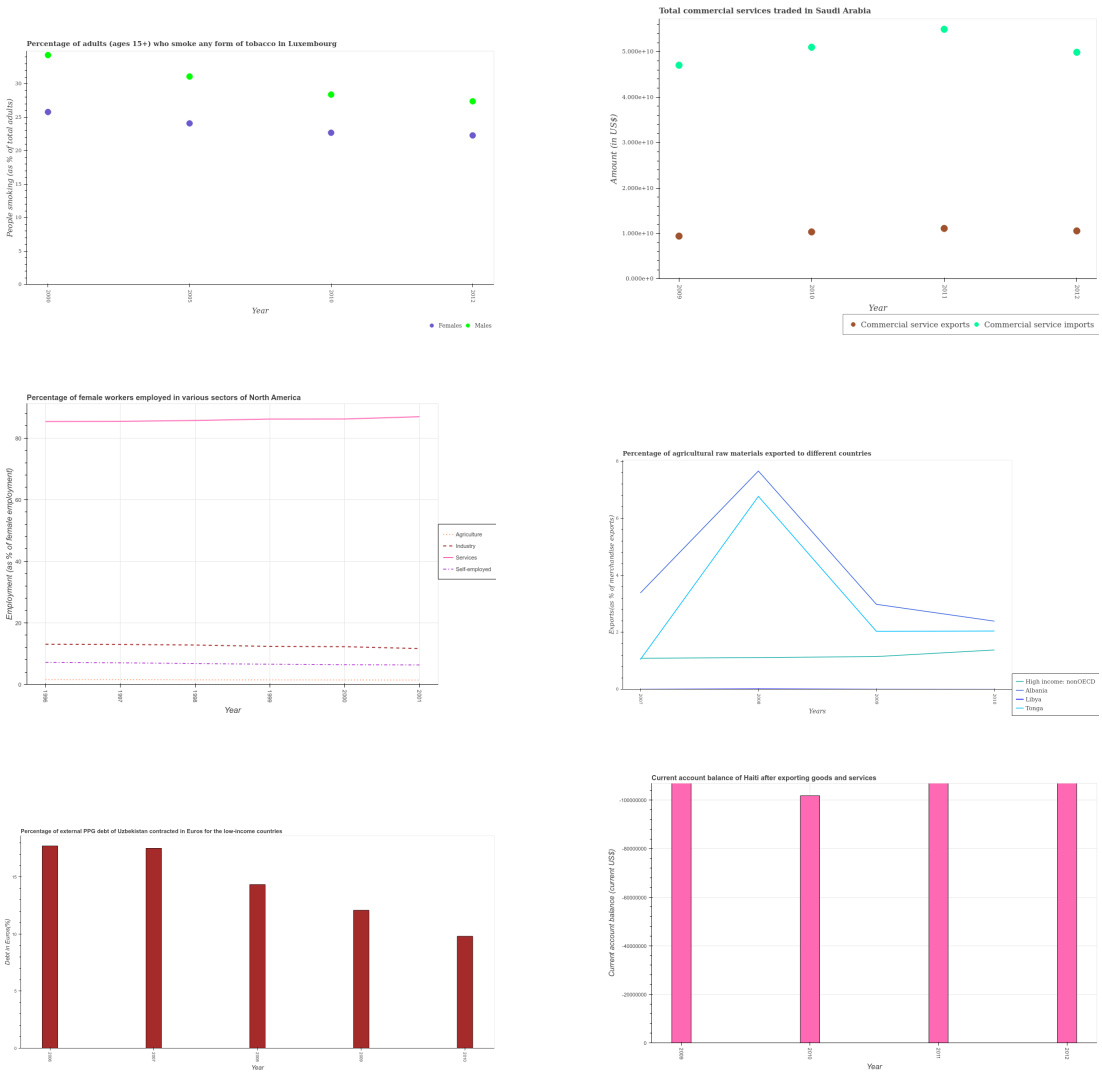


Figure A.2: Sample charts taken from the PlotQA dataset. Two scatter plots, two line plots, and two bar plots were chosen at random. Lack of variability is immediately evident in the visual styling and data distribution.

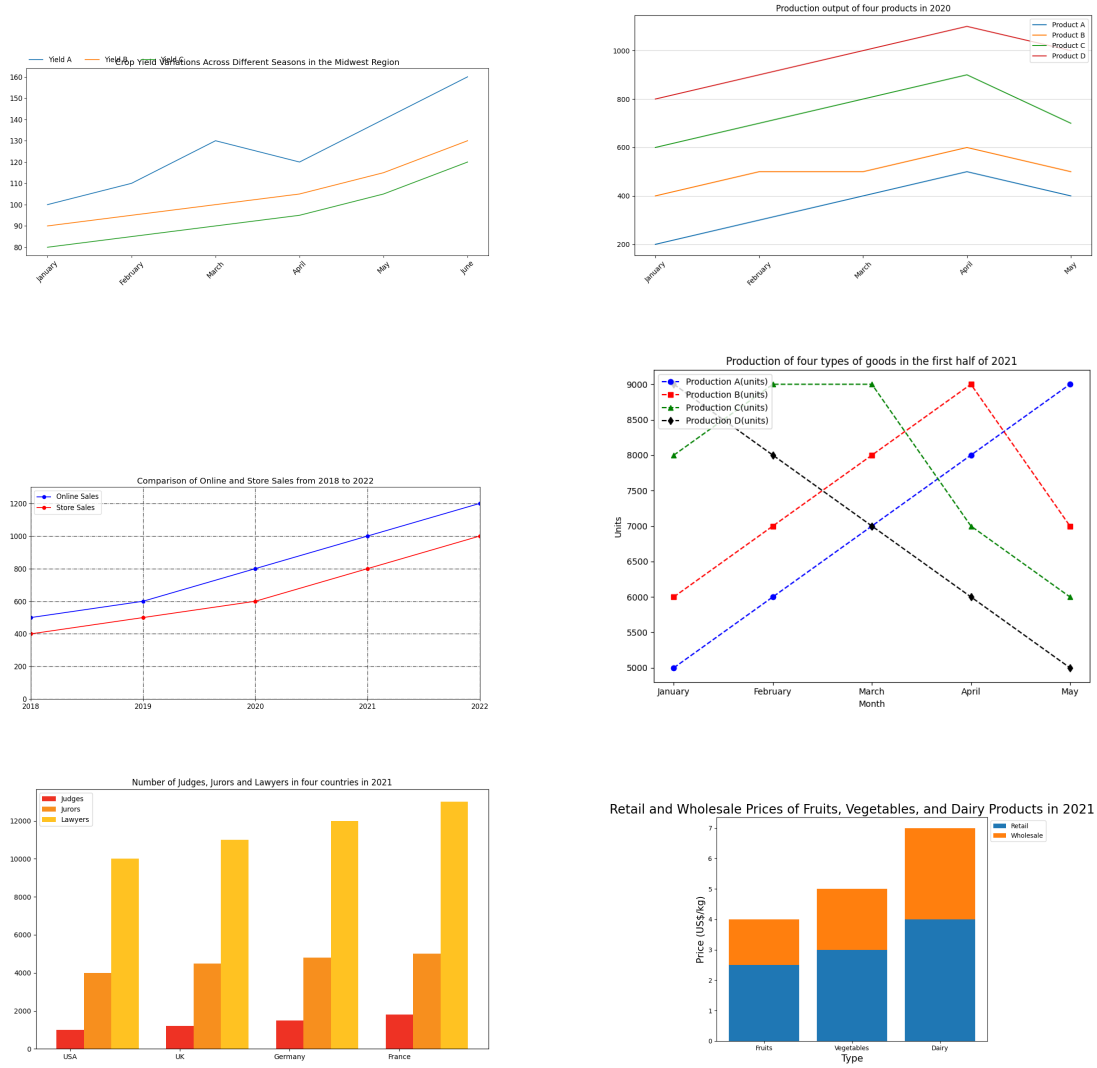


Figure A.3: Sample charts from the ChartX Dataset. There is significantly more variety in visual styling then in either ChartQA or PlotQA. Like the ChartQA dataset, ChartX does not include scatter plots.

Appendix B

Open Source Model Performance for Chart Recreation

I experimented with using the QWEN-VL-MAX, Intern-VL-Chat-1.5, and LlaVA-1.6 models as open source alternatives to the GPT-4o model. None of them were capable of performing the task competitively to GPT-4o. The LlaVa-1.6 model proved incapable of generating code without errors for most tasks, while the other two models produced extremely simplistic graphs. Below are some input-output pairs of graphs generated by the models:

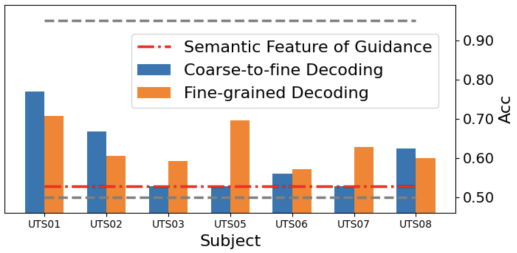


Figure B.1: Input chart taken from arXiv

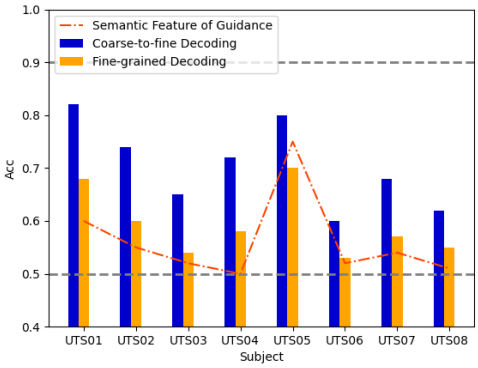


Figure B.2: GPT-4o Model Output (input chart shown in Figure B.2)

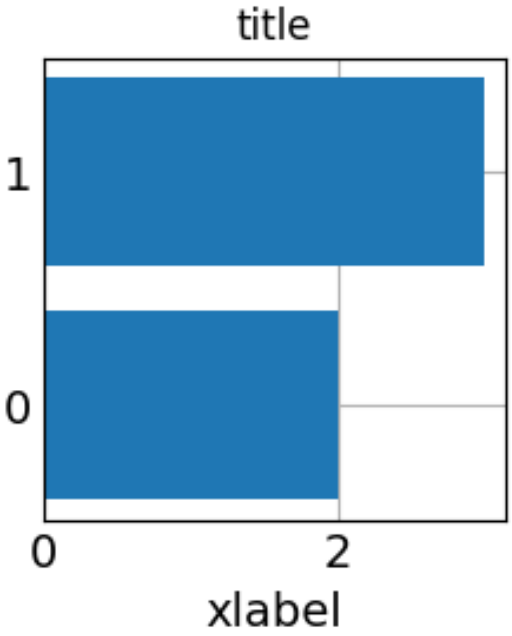


Figure B.3: QWEN-VL-MAX output (input chart shown in Figure B.2)

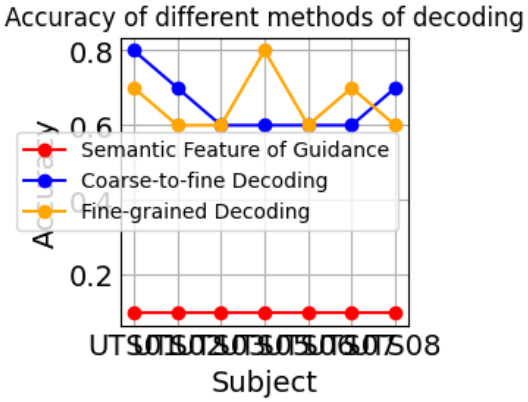


Figure B.4: Intern-VL-Chat-1.5 output (input chart shown in Figure B.2)

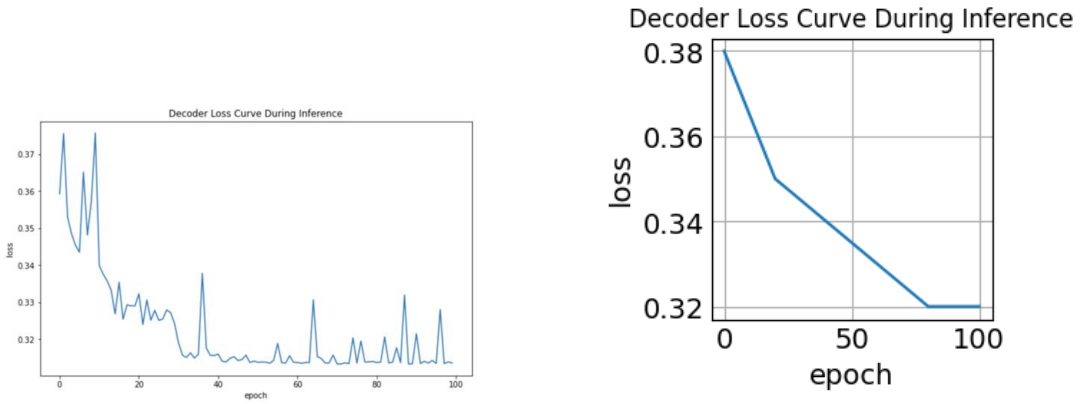


Figure B.5: Machine learning curve line chart scraped from arXiv (left) compared to Intern-VL-Chat-1.5 output (right).

Appendix C

BLIP Model testing

I curated a dataset by hand for testing different chart filtering procedures. The dataset contained 103 images into organized into 6 different categories, as detailed in Table C.1. Graph-like images were images that had visual features similar to graphs (sound waves, collections of colored points, images of sine waves against blank backgrounds). Graphs (no data) refers to graphs that I determined did not have data that could be meaningfully transcribed such as violin plots. The BLIP model achieved an impressive 96.1% accuracy on the assembled dataset, with just two false negatives and two false positives. Importantly, the false positive rate was very low at just 2.5%.

Clip Art	Diagram	Image	Graph-like Images	Graphs (no data)	Graphs
12	10	28	20	7	24

Table C.1: Sample Dataset for Testing Chart Filtering Techniques

As I mentioned in the methodology, I also experimented with a few other chart filtering techniques, but they were discarded before I had assembled the sample dataset for testing because of obvious poor performance and high false positive rates.

Appendix D

Image Filtering

As such, it was essential that we implement an effective filtering process with a low false positive rate (incorrectly classifying non-chart images as charts). Despite its use in the creation of the SCICAP and LineCap datasets, I determined that the 11% false positive rate of the FigureSeer classifier when detecting graphs was too high [30]. Initial attempts with other pre-trained classifiers also displayed high rates of false positive misclassifications [47, 48]. Larger image-caption alignment models such as the OpenAI CLIP model were similarly ineffective [49].

Interestingly, inclusion of the term "chart" when filtering with the BLIP model actually negatively impacted the performance of the model, perhaps because of the slightly different definitions of the term in academic and colloquial usage.

D.1 BLIP model

The BLIP model was rather effective at filtering out non-graph images. 94.0% of images into the BLIP model were classified as non-graphs and removed. Still, a large amount of non-graph images remained in the dataset. As you can see in Table D.1, the original composition of the dataset after BLIP filtering was roughly 73% graphs. While 73% is far from perfect, this is an impressive rate given the overwhelming majority of non-graph images in the input dataset, especially for the simplicity of my criteria (presence of either the term "graph" or the term "plot" in the caption). While I experimented with other filtering procedures to further reduce the amount of non-graph images, none of them were close to as successful as the BLIP model.

D.2 GPT-4o Filtering and Use of the NONE Response

ChatGPT-4o did not make use of the NONE response very frequently. When it did, it was generally for images that were not graphs at all that had slipped past the BLIP model. Table D.2 shows the distribution of image types that caused GPT to exercise this non-response option. This is a common pattern observed in large language models (LLMs), where the model's design encourages it to attempt solving tasks rather than opting out, even when presented with unfamiliar or irrelevant input. This tendency reflects the model's training, which prioritizes generating a response over recognizing situations where a non-response might be more appropriate [50, 51].

In Table D.2, Graph Like refers to images with visual characteristics very similar to those seen in graphs (e.g. a visual representation of a frequency wave that does not contain axes). Graph Part refers to an image containing a segment of a graph. These Graph Part images were very common in the input dataset, as the arXiv API sometimes downloaded images in small "strips." Some graphs could be divided into more than 100 of these segments, diluting the dataset with images that, although they featured all of the representative features of graphs, were not large enough to be able to be meaningfully recreated. Figure D.1 shows a couple examples of this problematic image type.

Category	**Graph**
True Graphs	73%
Natural Image	4%
Graph Like	9%
Graph Part	12%
Diagram	1%
Text	1%

Table D.1: Breakdown of frequency of different image types in the dataset of images after BLIP filtering. Percentages were calculated by randomly sampling 100 images and manually categorizing them into one of the six categories above.

Category	**NONE**
Input Images	5%
Natural Image	33%
Graph Like	6%
Graph Part	42%
Diagram	11%
Text	3%

Table D.2: Breakdown of frequency of different image types which caused the GPT-4o model to reply NONE. Percentages calculated using a sample size of 100, as in table **??**

Figure D.1: Examples of “Graph Part” images.

Appendix E

GPT-4o Prompting

E.1 Experimentation

I experimented with a number of different prompts for the code creation task. At first, I just messed around with different prompts in the web interface for ChatGPT-4o. Once I found a prompt that gave good results, I began experimenting with small batches on the API. One major difference between using the web interface for GPT-4o and the API is the ability to pass a system prompt as well as a user prompt. In other words, the user has the ability to first set the context and role of the AI, providing guidelines for how the model should act. This is an added layer of customization not available in the web interface where users are only able to input a single prompt and are not able to separately adjust this contextual layer. The image URL can then be input as another user prompt, separate from any textual prompts. Interestingly, after testing various combinations of system and user prompts, I found that the model performed best when the entirety of the prompt was input as a system prompt.

Some interesting findings I made that influenced my final prompt:

1. I experimented with dividing the contents of the prompt between the system prompt and the user prompt. I would provide a one sentence description of the context/situation to the AI ("You are a highly skilled...") in the system prompt and then detail the task in the user prompt. I found that the prompts that performed best had all of the textual prompt in the system prompt, with the user prompt serving only to provide the URL link to the image to analyze.

2. The best prompt was one which explicitly told GPT that it was a "highly skilled AI," rather than an "image analysis expert" or other human expert.

3. Prompts that listed out important elements that should be retained such as color scheme, line styles, markers, etc. gave the best results.

4. Although I explicitly asked GPT not to include other comments to limit token usage, it still included short comments in the majority of the responses.

5. I included the ability to respond "NONE" as a final check on whether the image input was actually a graph. This was because I had seen that the SalesForce BLIP model did sometimes mistakenly categorize images with graph like features (ex: images of sound waves, clip art of a sine wave) as graphs. I also gave it the option to return "NONE" if the graph was not suitable for transcription, but the model did not utilize this non-response option very frequently (just 9.9% of the time).

E.2 Prompts Given to the GPT-4o Web Interface

These are the prompts I experimented with using the GPT-4o web interface. I input graphs one at a time and compared input and output graphs. There is no system prompt included as I was interacting with the web interface.

Prompt 1

"Write code in matplotlib to generate a graph that looks as visually similar to this graph as possible. Return the code. If the image presented is not a graph from which data can be extracted, return NONE."

Prompt 2

"Generate matplotlib code to create a graph that visually resembles the provided image. If the image does not contain extractable data or cannot be manually digitized (e.g., lacks axes or displays patterns only), respond with the word NONE."

Prompt 3

"Generate matplotlib code to create a graph that visually resembles the provided image. In the first line, add a comment which best describes the graph type (bar, line, histogram, etc.). If the image does not contain extractable data or cannot be manually digitized (e.g., lacks axes or displays patterns only), respond with the word NONE."

Prompt 4

"Generate detailed matplotlib code to create a graph that closely matches the provided image. The code should run without errors when copied and pasted into a Jupyter notebook. Ensure the following:
Accurately interpret and replicate the axes, labels, and legends from the image. Match the data points, lines, and other graphical elements as precisely as possible. Do not include comments in the code. If the image does not contain extractable data or cannot be manually digitized (e.g., lacks axes or displays patterns only), respond with the word NONE."

E.3 Prompts Given to the API

The first prompt I tried actually turned out to be one of the best, and the final prompt I used turned out to be only a slight modification of Prompt 1.

Prompt 1

System: You are a highly skilled AI specialized in generating Python code for creating graphs that are visually similar to provided examples. Generate Python code to create a graph resembling the provided image. Ensure the graph type, data distribution, color schemes, line styles, markers, legends, labels, axes, error bars, and data markers match the example. Include a comment at the top with the graph type (e.g., # bar). Do not include any other comments. If the image does not contain transcribable data (e.g., lacks labeled axes, displays only patterns), respond with NONE.

User: Image URL

I then experimented with including a short system prompt explaining how I wanted the AI to behave (controlling its context), as seemed to be the function of the system prompt, and then including the actual request I was giving to the AI as the user prompt. These gave differing result qualities, but they were all on the whole less effective than Prompt 1.

Prompt 2

System: You are an AI specialized in understanding and reproducing graphs from images.

User: Review the image. If it is a graph with data that can be transcribed, provide Python code to visually recreate the graph. Include a comment at the top specifying the graph type (# e.g., # bar). Do not include any other comments in your code. If the image does not contain a graph, respond 'NONE'. If the graph lacks transcribable data, respond 'NO DATA'.

Prompt 3

System: You are an image analysis expert who excels in converting visual data into code.

User: Analyze the image. Identify if it is a graph with data that can be converted to a table. If so, generate Python code to replicate the graph. If it's not a graph, reply "NONE". If it's a graph without extractable data, reply "NO DATA". Include a comment at the top of the code specifying the graph type (# e.g., # bar). Do not include any other comments in your code.

Prompt 4

System: You excel at extracting data from graphical images and recreating them using Python.

User: Inspect the provided image. Determine if it contains a graph with transcribable data. If it does, generate Python code to recreate the graph as visually similar as possible. Include a comment at the top of the code specifying the graph type (# e.g., # bar). Do not include any other comments in your code. If it's not a graph, return "NONE". If the graph does not have transcribable data, return "NO DATA".

Prompt 5

System: You specialize in image-based data extraction and graph recreation using Python.

User: Please analyze the attached image. If it is a graph with data points that can be converted to a table, generate Python code to recreate it as visually similar as possible. If it's not a graph, respond "NONE". If the graph does not contain extractable data, respond "NO DATA". Include a comment at the top of the code specifying the graph type (# e.g., # bar). Do not include any other comments in your code.

Prompt 6

System: You are an AI expert in interpreting and reproducing visual data from images.

User: Evaluate this image. If it shows a graph with data that can be transcribed into a table, generate Python code to recreate the graph visually, including a comment at the top specifying the graph type (e.g., # histogram). Do not include any other comments in the code. If it is not a graph, reply "NONE". If it is a graph without transcribable data, reply "NO DATA".

Finally, in an effort to get the AI to return data along with the code, I experimented with prompts asking GPT-4o to return responses formatted as JSON with the graph type, data, and then the code all in separate fields. Surprisingly, this drastically reduced the quality of the output graphs, so I gave up on this way of getting the data for the code. On top of that, I also had concerns that the model might hallucinate and return data that was not the same as what was in the code. Without parsing the code, there would be no way to ensure that the data actually corresponded to the graph. As such, I quickly abandoned this strategy.

Prompt 7

System: You are an image analysis expert who excels at extracting tabular data from graphs and replotting them.

Finally, in an effort to get the AI to return data along with the code, I experimented with prompts asking GPT-4o to return responses formatted as JSON with the graph type, data, and then the code all in separate fields. Surprisingly, this drastically reduced the quality of the output graphs, so I gave up on this way of getting the data for the code. On top of that, I also had concerns that the model might hallucinate and return data that was not the same as what was in the code. Without parsing the code, there would be no way to ensure that the data actually corresponded to the graph. As such, I quickly abandoned this strategy.

User: Analyze the provided image and identify if it is a graph with data that can be converted to a table. If so, categorize the graph, extract the data into a table, and generate Python code to replicate the graph. Your response should be in the following format: { “graph_type”: “”, “data”: [# data in a JSON dictionary], “code”: # Your Python code here }. Do not include any comments in your code. Assume all necessary packages have been imported. If the provided image is not a graph, reply “NONE”. If it is a graph but it does not contain extractable data, reply “NO DATA”.

Prompt 8

System: You are an image analysis expert who excels at extracting tabular data from graphs and replotting them.

User: Analyze the provided image and identify if it is a graph with data that can be converted to a table. If so, categorize the graph, extract the data into a table, and generate Python code to replicate the graph. Ensure the graph type, data distribution, color schemes, line styles, markers, legends, labels, axes, error bars, and data markers match the provided graph. Your response should be in the following format: { “graph_type”: “”, “data”: [# data in a JSON dictionary], “code”: # Your Python code here }. Do not include any comments in your code. Assume all necessary packages have been imported. If the provided image is not a graph, reply “NONE”. If it is a graph but it does not contain extractable data, reply “NO DATA”.