参赛队员姓名：顾淇元

中学：上海市世外中学

省份：上海市

国家/地区：中国

指导教师姓名：Zhi-Qi Cheng

指导教师单位：Carnegie Mellon University

论文题目：ChartLLM: Unlocking the Multimodal Potential of LLMs for Chart Comprehension

ChartLLM: Unlocking the Multimodal Potential of LLMs for Chart Comprehension

Qiyuan Gu
Shanghai World Foreign Language Academy
phoenix_1203@icloud.com

Abstract

Charts are vital for explaining and communicating intricate data in research. Although Large Language Models (LLMs) like GPT-3.5/4 excel at text, their chart comprehension is inadequate. Despite claims about GPT-4's chart handling, its utility is constrained by lack of interfaces. Moreover, existing techniques often remain confined to narrow tasks, impeding adaptability. This predicament necessitates sophisticated, universally applicable chart comprehension techniques.

We propose ChartLLM, an end-to-end pipeline for chart data extraction and question-answering. Our specialized keypoint detection module employs recursive Hourglass networks across multiple scales to localize chart components. We unify diverse annotation formats into consistent keypoint representations. A Transformer encoder then models relationships for grouping. This facilitates interpretable structured data extraction. For question answering, we fine-tune T5 on the extracted data using special tokens to associate questions and answers.

Extensive experiments on 3 datasets demonstrate state-of-the-art accuracy exceeding 90% on FigureQA and DVQA, revealing consistent improvements over ChartQA's T5 approach. However, performance on ChartQA's real-world charts remains constrained by extraction limitations for complex visuals like dense line charts. We also instantiate our solution in a real-world application, substantially reducing manual analysis.

Our work marks significant progress toward universal chart comprehension. It also enables seamless integration with LLMs like T5, expanding their scope. Our code is publicly available on GitHub. This technique could profoundly enhance scientific communication and discovery. The source code and mode are available at https://github.com/phoeniiix1203/ChartLLM.

Keywords: *chart question answering, multimodal, large language model, data extraction, Transformer*

Contents

1. Introduction

Charts serve as important tools for research by explaining and communicating intricate data. However, despite their prowess in language tasks, Large Language Models (LLMs) like GPT-3.5/4 demonstrate inadequate chart comprehension abilities. Their text-centric architecture precludes processing visual data like charts. This deficiency severely limits their applicability, as charts have become ubiquitous across academia, business, and media for summarizing complex data. Moreover, ChartQA poses distinct challenges versus Visual QA on ordinary images, including out-of-vocabulary terms, demanding precision, and complex reasoning.

While charts are difficult for LLMs to understand, tables are more understandable for LLMs. As tabular representations of underlying data, tables organize information systematically across rows, columns, and headers. This explicit structure facilitates locating and reasoning about data to answer questions. Moreover, LLMs are already trained extensively on tasks involving structured text. Therefore, "derendering" charts into tables could enable capitalizing on innate LLM capabilities.

We introduce ChartLLM, an end-to-end pipeline for extracting chart data as tables and fine-tuning LLMs for question answering. Our specialized keypoint detection module employs Hourglass networks to localize components across scales. We unify annotations into consistent representations. Next, a Transformer encoder models relationships for grouping keypoints. This structured data extraction precludes predefined heuristics. For question answering, we fine-tune T5 on the extracted tables using special tokens.

Experiments on FigureQA, DVQA and ChartQA demonstrate over 90% accuracy on synthetic data but also limitations generalizing to complex real charts. We have instantiated our solution in an application, substantially reducing manual chart analysis. Our work marks significant progress in unlocking LLMs for multimodal comprehension. By effectively "derendering" charts into digestible tables, our technique finally unlocks the latent capability of LLMs for chart comprehension. It expands their scope beyond pure text, enhancing scientific communication and discovery. This work only scratches the surface of how LLMs could integrate vision and language - further multimodal expansions could profoundly augment their utility.

In addition, we have made our training, analysis, and visualization code publicly available on GitHub: https://github.com/phoeniiix1203/ChartLLM. We have also made a simple demonstration interface for our model, as shown in Fig. 1 and Fig. 2 below. It's worth noting that while the current implementation focuses on the three most common chart types—bar, line, and pie charts—the framework is designed to be easily extensible to accommodate new chart types, such as whisky charts, thereby enhancing its utility and applicability.

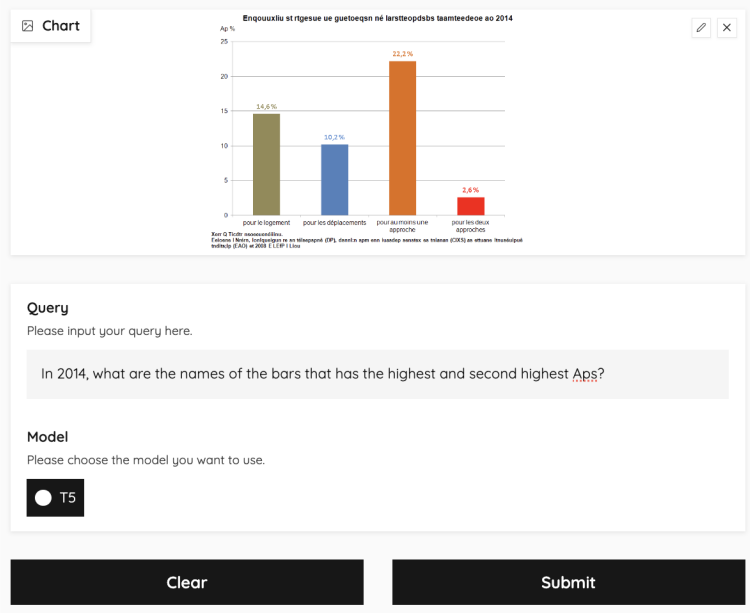

Figure 1. User input interface of demonstration

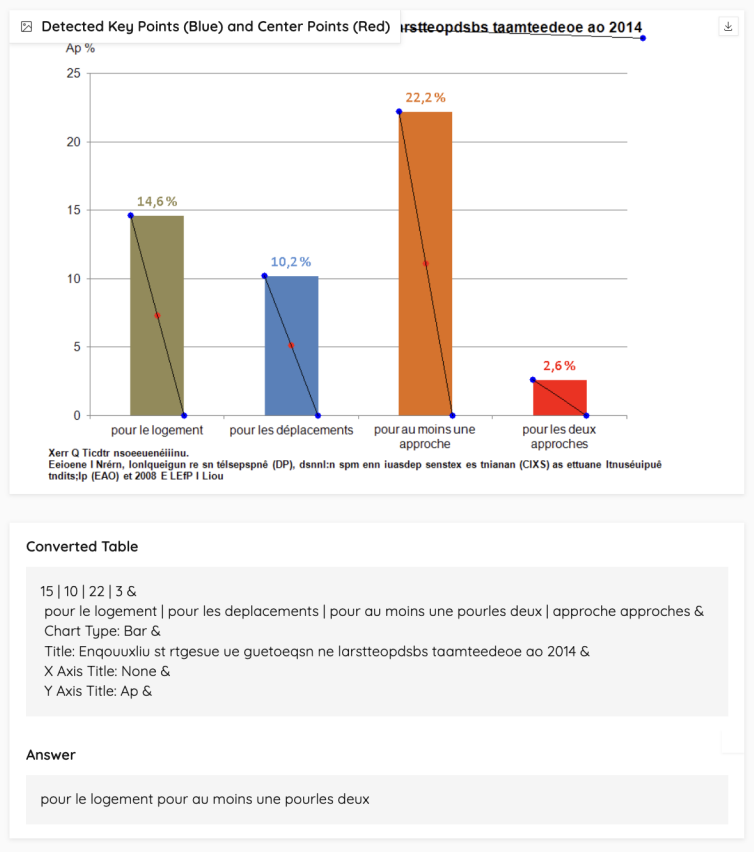

Figure 2. Output interface of demonstration

Overall, our research offers the following four significant contributions:

1. **End-to-End Framework for Chart Data Extraction and LLM Enhancement:** We introduce a groundbreaking end-to-end framework tailored for extracting chart data and refining LLMs to adeptly handle question-answering tasks. This innovative approach facilitates efficient interpretation and analysis of visual data representations.

2. **Specialized Keypoint Detection and Relation Modeling:** Our methodology incorporates a unique keypoint detection module leveraging Hourglass networks in tandem with a Transformer encoder. Together, these technologies ensure precise and unified chart component localization, and they sophisticatedly model relationships between keypoints. This design eliminates the dependence on predefined heuristics.

3. **Empirical Performance and Practical Implementation:** Our framework exhibits remarkable accuracy in chart interpretation, achieving over 90% on synthetic datasets such as FigureQA and DVQA. Furthermore, we have successfully implemented our solution in real-world applications, markedly reducing the demand for manual chart interpretation.

4. **Insight into Future Directions:** Committed to propelling the field forward, we have made our code publicly available, promoting community collaboration and aiding future research. We also offer an exhaustive analysis of the existing limitations of our approach and outline avenues for potential improvements. This transparency aims to spur further advancements in multimodal comprehension using LLMs.

In summary, while this work takes the crucial first step of equipping LLMs with basic chart comprehension skills, there remain extensive opportunities to build on this foundation across extraction, reasoning, generation, evaluation, and architecture. Fulfilling the immense promise of multimodal LLMs will require continuing this trajectory. We are excited and optimistic about future innovation in this direction.

2. Related Work

This section provides an exhaustive review of the literature pertinent to the domain of chart interpretation and understanding. Specifically, this review encompasses seminal works focused on a multitude of areas including, but not limited to, chart-based question-answering (Chart-QA), chart-to-table conversion, component detection within charts, table-based question-answering, and the role of large language models in these contexts.

2.1. Chart-QA Task

While some previous approaches have attempted to tackle the Chart Question Answering (ChartQA) task, their adaptability to various types of chart data remains a challenge: Methods by Kahou et al. [2] and Kafle et al. [5] primarily focus on employing convolutional neural networks (CNN) and long short-term memory (LSTM) architectures. These techniques excel at visual and simple temporal data processing but are less effective in dealing with intricate linguistic aspects in charts and does not support open-vocabulary question-answering on charts. As a result, end-users are constrained to posing questions based on a narrow set of pre-defined templates, thus significantly impeding the practical applicability of these systems. Alternatively, the PreFIL model with two parallel Q+I fusion branches [6] has also emerged as a sophisticated approach for handling both text and visual elements. However, its complexity makes it less straightforward to integrate as a modular extension to existing language models, especially for more generalizable solutions. *In light of these challenges, we advocate for the development of a hybrid methodology that can convert charts into a format more conducive to interpretation by large language models. Such an approach would amalgamate the specialized techniques seen in ChartQA systems with the expansive natural language processing capabilities intrinsic to large pre-trained models.*

2.2. Table-QA Task

In the initial stages of research, weakly-supervised semantic parsing for table-based question-answering (Table QA) largely relied on hand-crafted features and prescriptive grammar rules [4]. Subsequently, extractive methodologies emerged, focusing on the direct selection of token spans from linearized tables as potential answers or evidential material. Notable among these is the TAPAS model by Herzig et al. [3], which adapts the Transformer architecture for table-centric tasks. Similarly, the work by Yin et al. utilizes a neural programmer to interpret queries and generate responses based on tabular data. In a more recent development, large language models such as T5 [13] have been tailored for Table QA. These adaptations employ specialized training paradigms that integrate table structures into the model's understanding, thereby yielding promising results [12]. *These advances indicate transformer-based architecture, including large language models, can excel in interpreting and querying tabular data.*

2.3. Chart-to-Table Task

One compelling avenue for implementing the aforementioned hybrid approach involves transforming Chart-QA tasks into Table-QA tasks. However, existing efforts in this direction are fraught with inherent limitations. Kim et al. [7] were among the pioneers in this domain. They convert chart questions into table-based questions, leveraging the capabilities of the Sempre TableQA algorithm. This approach is valuable but highly limiting, as it assumes the availability of the underlying table for each chart, making it less practical for real-world applications where such tables are often missing.

To address this, Methani et al. [10] and Masry et al. [9] employ techniques like Faster R-CNN and Tesseract OCR to extract tabular data directly from charts. They further utilize TableQA models such as T5 and TAPAS for processing. While more versatile, these approaches are tailored for specific chart types—like bar, line, and pie charts—and rely on heuristic methods. This fragmentation requires separate training models for each chart type, thereby complicating the model architecture and operational processes. *We argue that the limitations inherent in existing methodologies impede the advancement of a unified, flexible framework capable of comprehensively understanding charts. To address this lacuna, our research concentrates on leveraging diverse chart data types to train a model that autonomously learns to identify various chart components. Importantly, our approach obviates the need for pre-defined heuristic rules for differentiating among chart components.*

2.4. SOTA Detection Methods

This subsection delves into contemporary techniques for detecting chart components, a critical phase in the conversion of charts to tables. These methodologies can be broadly categorized into two paradigms: bounding box detection and key-point detection.

(1) Bounding Box Detection. The initial, and perhaps more intuitive, methodology is bounding box detection. In this approach, rectangular boxes are superimposed on objects within an image to demarcate their spatial location and scale.

These bounding boxes can either be manually annotated, as exemplified in studies like [2], or algorithmically generated through techniques utilized in research such as [5], [10], and [14]. While simple to understand, bounding boxes are sensitive to scale and aspect ratio. Therefore, multiple chart component classes that may appear in various sizes and orientations make creating bounding box rules that effectively isolate each class complex and time-consuming. In addition, for components with irregular shapes, such as the pie parts in pie graphs, a bounding box can be an inefficient representation because it may include a lot of "empty" space that doesn't contain the object.

(2) Key-point Detection.On the contrary, our research adopts the second prevalent methodology, which is key-point-based object detection. This methodology involves identifying specific points in the object that describe its features or geometry. Studies like [8] and [9] have also employed this technique. The key-point-based methodology offers several advantages. Primarily, it reduced the need for object-specific rules, as once key points are identified, they can often be used universally across similar types of objects. For example, if the key points for one type of bar chart are identified, those key points might be applicable to other types of bar-like objects in charts without many modifications. Secondly, the key-point-based approach can potentially improve the accuracy of element detection, as key points often encapsulate critical information about the data elements they represent. Lastly, the key-point-based approach is slightly more computationally efficient, as it focuses on fewer, more meaningful points rather than scanning and processing large bounding areas.

In our work, we employ Hourglass networks [11] to detect key points. Hourglass networks have demonstrated efficacy in key-point detection tasks, notably in human pose estimation. The symmetric, multi-scale, top-down design of hourglass networks allows them to capture both global and local features effectively. This makes them well-suited for detecting key points in charts, where understanding the relationship between local elements and the overall structure is crucial.

2.5. Large Language Models

Large language models like GPT (Generative Pre-trained Transformer) [1] and T5 (Text-To-Text Transfer Transformer) [13] are neural network based models trained on extensive textual datasets. These models typically feature an architecture based on the Transformer, which was initially introduced by Vaswani et al [15]. The Transformer architecture is composed of multiple layers of self-attention mechanisms and feed-forward neural networks, facilitating the efficient learning of long-term dependencies in text. GPT models utilize a decoder-only architecture, while T5 employs an encoder-decoder framework that allows it to be fine-tuned for a broader range of tasks. Despite their complexity and capabilities, these models are fundamentally designed to process text. They lack native modules for handling non-textual data types like images or audio.

3. Our Porposed ChartLLM

In this section, we first delineate the procedures for dataset amalgamation and data format transformation to optimize them for our training tasks. Subsequently, we delve into the specifics of our key-point detection and grouping strategies for chart component identification. Following this, we elaborate on the design of an Optical Character Recognition (OCR) module, proposed for the extraction of textual elements from charts. This OCR module is then integrated with previously identified chart components, forming a cohesive system to de-render charts into table formats. Finally, we discuss the deployment of large language models in conjunction with the generated tables to facilitate the chart question-answering task.

3.1. Data Preparation

We opt to utilize the ExcelChart400K dataset as the foundational basis for our training data, a dataset originally introduced by the authors of ChartOCR [8]. ExcelChart400K contains chart images sourced by crawling publicly available Excel sheets on the internet. This data collection strategy ensures that the dataset closely mirrors charts as they are used in real-world scenarios, lending a high degree of practical applicability to our model.

However, the structure of the original data set is problematic for our training process. ExcelChart400K is organized into separate subsets, each specifically tailored for bar, line, pie charts. This format was originally designed to train distinct, type-specific information extraction models. However, such a structure is not conducive to the training of our unified model, which aims to handle multiple types of charts simultaneously.

To rectify this, we undertook several modifications to the dataset:

(1) Data Filtering. Firstly, we scrutinized the dataset to identify images that exclusively contain annotations for primary chart elements—like bars, lines, or pies—but lack annotations for other crucial components such as titles and axis labels. While these images may be useful for training specialized detection models, they are not suitable for our holistic approach. As a result, we pruned these images from our dataset. After this filtering process, our revised dataset comprises 115,776 images for training, 3,695 images for validation, and 4,078 images for testing.

(2) Annotation Relabeling. Next, we restructured the annotation system to clearly distinguish between different types of chart components. In the original dataset, the 'category id' was uniformly set to zero for all component types. To address this, we introduced custom category labels ranging from 1 to 7, each uniquely identifying a specific type of chart component. These labels are detailed in Table Tab. 3 in the Appendix. Note that in the original dataset, there exist two additional categories termed 'inner/outer areas.' As these two categories are not pertinent to our model's scope, we have opted to remove any annotations related to them for enhanced clarity.

(3) Unifying Representation. In the ExcelChart400K dataset, the annotations associated with different types of chart components exhibit notable inconsistencies and are tailored to the specificities of each chart type. For example, in line charts, the annotations are delineated as data points constituting the line, while in pie charts, they consist of a center point accompanied by two edge points that define a particular sector. In the case of bar charts and similar types of visualizations, the annotations are rendered as bounding boxes, specifying both the x and y coordinates of the top-left corner, as well as the dimensions of the box in terms of its width and height. Given these discrepancies, it becomes imperative to harmonize these annotations into a standardized format. Our approach focuses on converting these disparate annotation types into a unified representation, which consistently specifies key points and center points for each chart component.

To unify these disparate annotation formats, we define the pivot points on the lines in line charts, the center points plus the intersection points on the arc of pies in pie charts, and the upper left and lower right corners of other types of components as key points p_k. We also define the position centers of all kinds of components as center points p_c. As shown in Fig. 3, Fig. 4 and Fig. 5, the blue dots represent the key points, and the red dots represent the center points. The black line joining key points and center points shows that they belong to the same component. Note that under such definition, a single key point often forms a group with different center points simultaneously. For instance, as shown in Fig. 4 and Fig. 5, the same key point may be grouped with adjacent center points on either side.

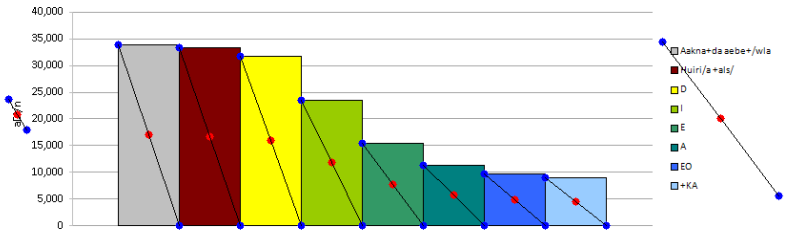

Figure 3. Key points and center points in bar chart. **Blue:** Key Points. **Red:** Center Points.

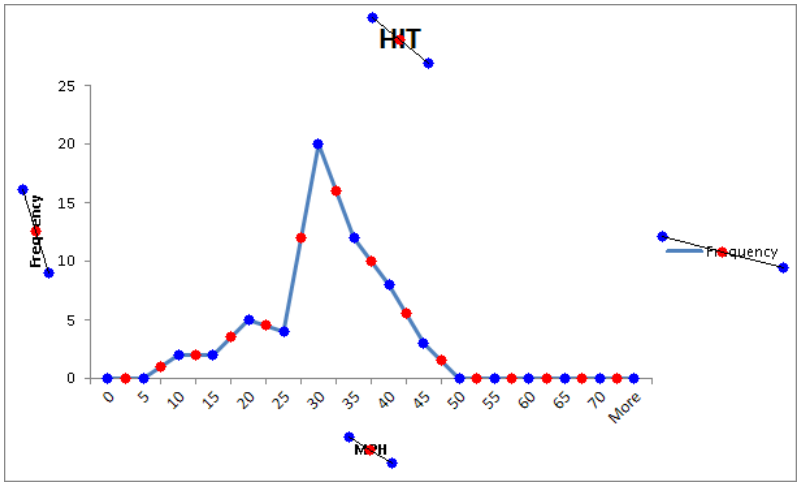

Figure 4. Key points and center points in line chart. **Blue:** Key Points. **Red:** Center Points.

Given these definitions, the conversion from original annotations to this unified center/key point format is straightforward. For bars in bar charts and the three title components, the coordinate of the center point is calculated by averaging the

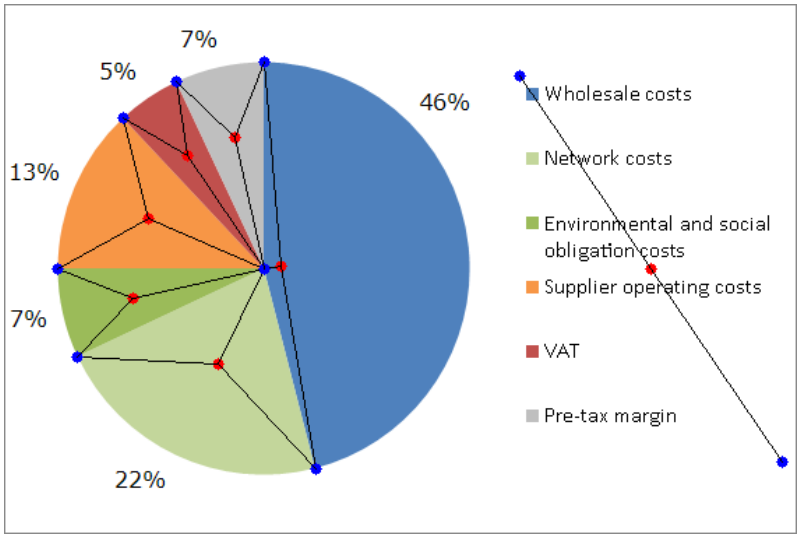

Figure 5. Key points and center points in the pie chart. **Blue:** Key Points. **Red:** Center Points.

coordinate of the top left and bottom right key points. For lines in line charts, the coordinate of the center point is calculated by averaging the coordinate of the starting point and the coordinate of the ending point of each line segment. For pies in pie charts, the calculation of coordinates of the center point is a bit more complex, we define a function called `get_center`, which takes as input the coordinates of three points a, b, and c. These points correspond to the center of the pie sector and its two edges respectively.

Algorithm 1 Calculate Center of a Triangle

procedure GET_CENTER(a, b, c)
  ca $\leftarrow [c[0] - a[0], c[1] - a[1]]$
  cb $\leftarrow [c[0] - b[0], c[1] - b[1]]$
  cross_z $\leftarrow$ ca$[0] \times$ cb$[1] -$ ca$[1] \times$ cb$[0]$
  if cross_z ≥ 0 **then**
    center$_x \leftarrow \frac{a[0]+b[0]+c[0]}{3}$
    center$_y \leftarrow \frac{a[1]+b[1]+c[1]}{3}$
    return (center$_x$, center$_y$)
  else
    center$_x \leftarrow 2 \times c[0] - \frac{a[0]+b[0]+c[0]}{3}$
    center$_y \leftarrow 2 \times c[1] - \frac{a[1]+b[1]+c[1]}{3}$
    return (center$_x$, center$_y$)
  end if
end procedure

The function begins by calculating two vectors, $\vec{ca}$ and $\vec{cb}$, which represent the vectors from point a to point c and from point b to point c respectively.

$$\vec{ca} = [c_x - a_x, c_y - a_y]$$
$$\vec{cb} = [c_x - b_x, c_y - b_y]$$

Next, we compute the cross product $\vec{ca} \times \vec{cb}$. The sign of this cross product indicates the direction of the angle between $\vec{ca}$ and $\vec{cb}$.

$$\vec{ca} \times \vec{cb} = ca_x \cdot cb_y - ca_y \cdot cb_x$$

If $\vec{ca} \times \vec{cb}$ is non-negative, we compute the centroid of the triangle formed by a, b, and c as follows:

$$\text{Center} = \left(\frac{a_x + b_x + c_x}{3}, \frac{a_y + b_y + c_y}{3} \right)$$

Otherwise, we choose another point as the center, calculated as:

$$\text{Alternate Center} = \left(2c_x - \frac{a_x + b_x + c_x}{3}, 2c_y - \frac{a_y + b_y + c_y}{3} \right)$$

In addition, for lines in line charts, we introduce an additional step to ensure uniformity. We identify the line with the maximum length in our dataset and pad all other lines to match this length. This normalization is critical for maintaining consistency when applying algorithms sensitive to input dimensions.

After transforming the annotation into the key point and center point format, the next step in our pipeline is data augmentation, which improves the generalizability and robustness of the model. We initially considered employing random cropping, a technique that extracts a smaller, random section from an original image, and using this "crop" for training, as part of our data sampling process. However, we observed that this approach was unsuitable for line charts. Random cropping led to the elimination of a significant number of data points in lines, introducing errors in subsequent steps. To compensate for the absence of random cropping, we employed Gaussian bump. Gaussian bump is a technique that involves modifying the intensity of data points based on a Gaussian distribution. This means that, unlike random cropping, Gaussian bump does not alter the position of key points but rather introduces variations in their attributes, allowing the model to generalize better without losing critical information.

3.2. Center/Key Point Detection

(1) Preliminary Stage of Center/Key Point Detection. In the initial stage of key point detection, we first input the image into a preprocessing layer to obtain an intermediate representation, denoted as inter. This preprocessing layer consists of two primary components: a 7×7 convolutional layer and a residual block. Both components have a stride of 2, meaning they skip over input data with a step size of 2. The convolutional layer has 128 output channels, facilitating multi-dimensional feature extraction from the input image. This configuration aims to reduce the image dimensions while retaining sufficient relevant information.

(2) Specialized Module for Key Point Detection. Next, we design a specialized module for keypoint detection, as shown in Figure 6. This module employs a five-level recursive structure, enabling the module to identify key points at different scales and levels of detail. Within this module, the input x is passed through an upsampling layer and a max-pooling layer, producing up1 and max1, respectively.

up1 has a larger feature map size, which aids in retaining more spatial details, thereby improving key point localization accuracy. max1, with a smaller feature map size, facilitates the extraction of global features, enhancing the model's robustness to key point detection under various conditions.

Then, max1 undergoes a feature transformation through an Hourglass layer, generating low1. The Hourglass layer is a symmetrical structure made up of multiple residual modules. It allows multi-scale feature fusion while maintaining high image resolution, effectively capturing the structural information of the target.

The recursive structure comes into play here. If the bottom level of recursion is reached, low1 is directly transformed into low2 through another Hourglass layer. Otherwise, low1 proceeds to the next level of recursion, using the return value of the next level as low2.

Subsequently, low2 undergoes another feature transformation through a reverse Hourglass layer, generating low3. Unlike the standard Hourglass layer, the reverse Hourglass layer is an asymmetric structure. It is still made up of multiple residual modules, but its design aim is to guide high-resolution local feature extraction using low-resolution global features.

The final step involves feature fusion of up1 and low3 to produce the final output. This fusion operation enhances the diversity and expressiveness of features, further improving the performance of the key point detection module.

(3) Formal Stage of Center/Key Point Detection. We observed that previous ChartOCR models employed a one-dimensional convolution layer and a pooling layer to further process features detected by the Hourglass Network. Specifically, for bar charts, ChartOCR used a center pooling layer, while for pie charts and line graphs, a corner pooling layer was utilized. However, our experiments showed that the corner pooling layer performed poorly for pie charts without obvious corner points. To unify the detection across the three types of charts, we discarded these pooling layers in our model.

In the formal stage of key point detection, we first initialize an empty list, outs, to store various outputs. During two iterations, several key steps occur: we first input the intermediate representation (known as inter) into the aforementioned

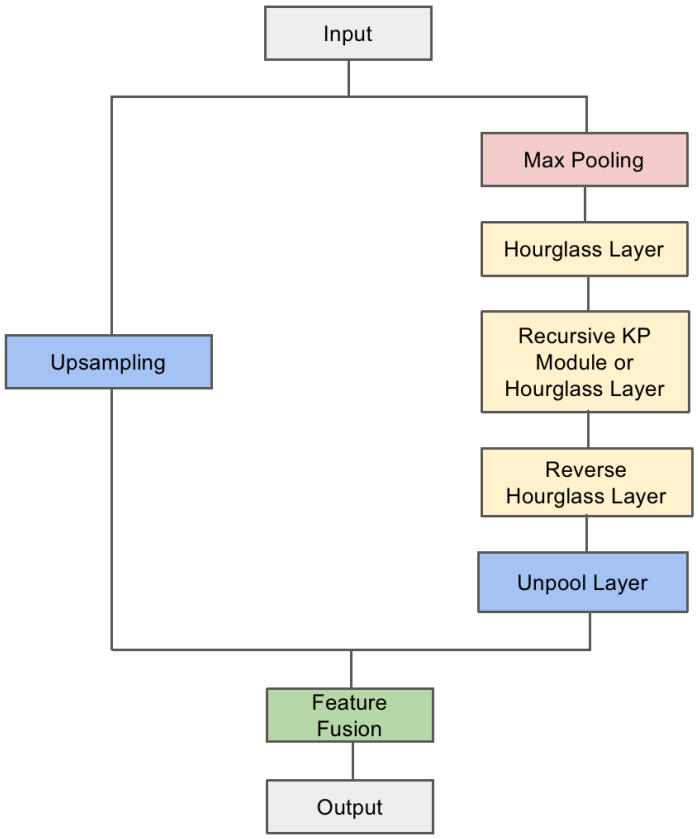

Figure 6. Self-defined Key Point/Center Point Detection Module

key point detection module. This module includes a detection layer for computing key points, denoted as kp. Next, we apply a convolutional layer to kp, obtaining a new representation, cnv. We then apply specialized key point and center point convolutions to cnv, generating key_cnv and cnter_cnv, respectively. For both key_cnv and cnter_cnv, we compute key_heat, center_heat, key_regr, and center_regr through center point and key point heatmap layers, as well as center point and key point regression layers. Subsequently, we transpose and extract features from key_regr and center_regr. Finally, these values are added to the outs list.

If the current iteration is not the final one, we need to update inter. We process inter and cnv through separate convolutional and batch normalization layers. The processed inter and cnv are then concatenated and further processed through a ReLU activation and a residual block to update inter. Through this detailed sequence of steps, we ultimately obtain feature maps of the same size as the original image. Each pixel value in each feature map represents the probability of a center point/key point existing at the corresponding image location.

(4) Loss Functions for Center/Key Point Detection. We combine Smooth L1 Loss and Focal Loss as our loss function for keypoint detection. Each of these loss functions addresses specific challenges: Focal Loss deals with class imbalance in classification tasks, while Smooth L1 Loss is used for regression tasks. Combining them allows us to leverage the benefits of both. Addition is the most common way to combine different loss functions, primarily because the additive operation maintains differentiability—a requirement for optimization algorithms like gradient descent. Additionally, it is also easier to analyze and understand compared to other more complex operations.

In object detection and segmentation tasks, there is often a significant imbalance between negative samples (i.e., background or non-target objects) and positive samples (i.e., target objects). This imbalance may lead the model to allocate more computational resources to learning the negative samples, thereby neglecting the fewer but more important positive samples. To address this, we employ Focal Loss, a modified version of the standard cross-entropy loss with an added modulation factor. This factor decreases as the model's confidence in classifying a sample increases, thereby mitigating the class imbalance

issue in object detection tasks.

The mathematical expression for Focal Loss is:

$$\text{Focal Loss} = -\alpha(1-p)^{\gamma}\log(p)$$

where p is the model's prediction, and α and γ are hyperparameters (set to 4 and 2 in this study, respectively) that control the allocation of weights.

Smooth L1 Loss is designed to alleviate some of the limitations of both L1 loss ($|x|$) and L2 loss (x^2). Its mathematical expression, where x is the difference between the predicted and actual values, is:

$$\text{Smooth L1 Loss}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

When prediction errors are very small, the gradient of the L2 loss may become exceedingly large, leading to model instability. L1 loss is non-differentiable at zero, which can cause issues during optimization. Smooth L1 Loss mitigates these issues by employing a squared term near zero, maintaining differentiate without being as insensitive to outliers as pure L1 loss.

3.3. Center/Key Point Grouping

In contrast to traditional chart recognition models such as ChartOCR, which employ a variety of complex heuristic rules to group detected key points, our research directly utilizes neural networks to predict the relationship between key points and center points. This approach eliminates the influence of human factors during the grouping stage, providing a more unified and accurate framework for chart information extraction.

In terms of model architecture, we first pre-train our model on the task of keypoint detection and subsequently fine-tune it for key point grouping. This combined approach of pre-training and fine-tuning is superior to training a model that simultaneously performs both key point detection and grouping. Through pre-training, the model learns general features for keypoint detection that can easily transfer to the task of key point grouping, thereby enhancing the model's overall performance.

Specifically, the initial phase of our model aligns with the key point detection model previously described. The image first undergoes preprocessing and passes through an Hourglass network module to generate feature maps for key points and center points

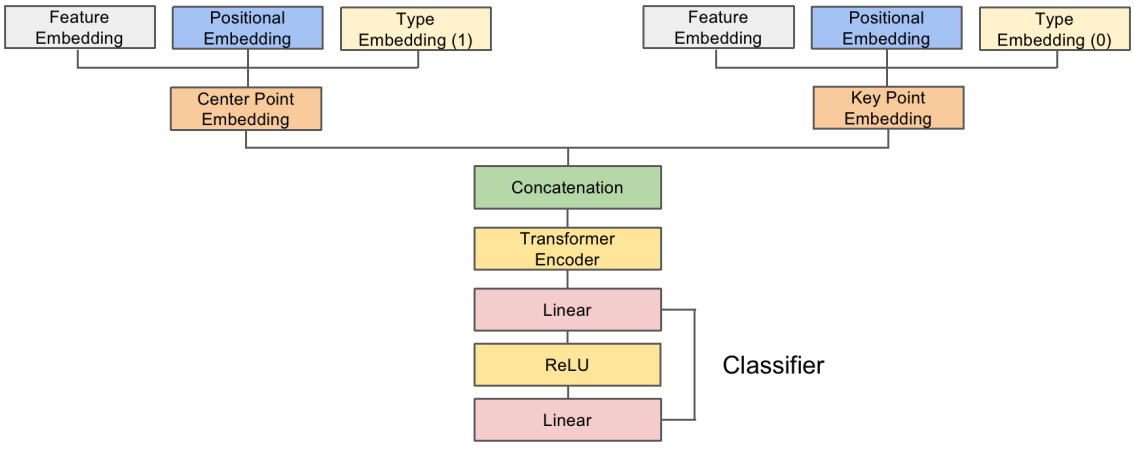

Figure 7. Key Point/Center Point Grouping Model (Above Structures are the Same as Key Point/Center Point Detection Model)

(1) Embedding Vectors for Key Points and Center Points. After obtaining the feature maps, as shown in Fig. 7, we further enrich the data by forming embedding vectors. Taking the center points as an example, we first extract their features from a specific batch to create feature embeddings. Next, based on the index of each center point, we compute its precise position within the image to generate position embeddings. Subsequently, we create a type embedding filled with ones to indicate that

these are center points. Ultimately, these three types of embeddings—feature, position, and type—are concatenated to form a complete embedding vector. The treatment for key points is similar, except that the type tensor is filled with zeros in this case.

(2) Transformer Encoder for Grouping. These embedding vectors are then fed into a Transformer encoder consisting of six layers, each having 260 model dimensions, four attention heads, and 1024 feed-forward network dimensions. The final step involves making predictions through a classification layer. In this layer, the 260-dimensional input is first transformed into 64 dimensions via a linear layer. This is then followed by a non-linear activation using a ReLU function. Finally, through a second linear layer, we obtain a 2-dimensional output that serves as the prediction for whether the points belong to the same group or not.

Given that the majority of the code for the center/key point detection model is subsumed within the center/key point grouping model, we have opted to economize on page length by only including the backbone of the center/key point grouping model in the Appendix.

(3) Loss Functions for Center/Key Point Grouping. During the training process, we employ the same two types of loss functions used in the key point detection phase to calculate the loss at the detection stage. These losses are then combined with the cross-entropy loss from the grouping results to form a composite loss function for the key point grouping task.

Cross-entropy loss is commonly used for classification problems. It measures the 'distance' between two probability distributions: the distribution predicted by the model and the actual 'ground truth' distribution. If the model's predictions are accurate, the cross-entropy loss will approach zero. Conversely, if the model's predictions are far from the true labels, the cross-entropy loss will be significant.

Assuming there are N samples and C classes, if the model predict that the i^{th} sample belongs to the j^{th} class is p_{ij}, and the true labels are represented by y_{ij} (where $y_{ij} = 1$ indicates that sample i belongs to class j, and $y_{ij} = 0$ otherwise), the cross-entropy loss is defined as:

$$\text{Cross Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij}) + (1-y_{ij})\log(1-p_{ij})$$

3.4. Integration of OCR

For the extraction of textual information within the charts, specifically for the titles of the value axis, the overall chart title, and the titles of the category axis (categories 5-7), we employ TesseractOCR version 5.3.0 (https://github.com/tesseract-ocr/tesseract), with the Page Segmentation Mode (psm) parameter set to 12. This particular setting is employed to indicate that the image consists of text with sparse regions.

Notably, prior to text extraction, we use the OpenCV library (cv2) to condition the image for optimal text extraction. The initial transformation to grayscale via the cv2.cvtColor function simplifies the image, thereby reducing computational complexity while retaining the essential features necessary for text recognition. This is followed by the application of Otsu's thresholding method using the cv2.threshold function to generate a binary image. The thresholding serves to differentiate textual elements from the background, essentially segmenting the regions of interest. Finally, we employ Fast Non-Local Means Denoising through the cv2.fastNlMeansDenoising function to reduce noise artifacts in the image. This denoising step enhances the clarity of textual elements, mitigating the likelihood of OCR errors. These preprocessing steps collectively serve to improve the quality of the input image, thereby facilitating more accurate and efficient text extraction using TesseractOCR.

3.5. Table Formation

Finally, we utilize the information obtained from both the component detection and OCR part to form the underlying table of the chart image. To obtain the data values in the chart, we employ the algorithm described in ChartOCR. This algorithm capitalizes on the consistent observation that y-axis numerical values are exclusively positioned on the left-hand side of the chart. Leveraging this observation, we compute the value represented by each chart component using their respective coordinates in conjunction with the coordinates of the y-axis numerical values.

The first segment of the resulting table contains the information on the independent variable depicted in the chart, with each data point separated by a vertical bar ("|"). Following this, the dependent variable(s) are also recorded, again separated by vertical bars.

To enrich the table with more contextual information, we incorporate additional metadata. This includes the type of the chart, which could be bar, line, or pie, the title of the chart, and the titles of the X and Y axes.

It is noteworthy that if a component is undetected during the process, the corresponding table value will be registered as `None`. An example of the obtained data table of Fig. 3 is shown in the Appendix.

3.6. Chart Question Answering

In the final question-answering on the table extracted step, we fine-tune the large pre-trained network T5 [13] following the similar practice described in the orginal ChartQA paper.

To elaborate on the specifics, the input fed into the encoder is formatted in a sequential manner, beginning with a `[CLS]` token, followed by the question tokens, and terminating with a `[SEP]` token and the chart content tokens. Similarly, the input for the decoder is structured to commence with a `[CLS]` token, succeeded by the answer tokens, and concluding with a `[SEP]` token and chart content tokens, as illustrated in Fig. 8. The architecture is configured to directly generate answers based on this input format.

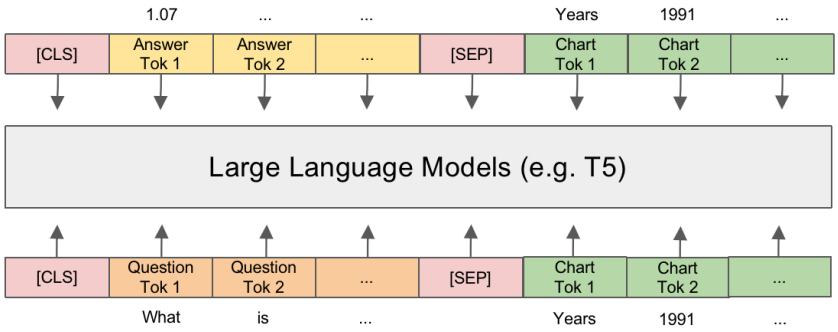

Figure 8. Fine-tuning large language models using table obtained

4. Evaluation

We fine-tune and evaluate ChartLLM on three existing datasets, Tab. 1 compares these datasets. These datasets have been chosen to represent a broad spectrum of complexity and real-world relevance. Early datasets include FigureQA [2] and DVQA [5], both contain a large corpus of synthetic data. FigureQA features approximately 100,000 charts along with 1.6 million questions. The questions are generated using 15 predefined templates and generally elicit yes/no answers. DVQA contains about 300,000 charts and a staggering 3.4 million questions. These questions are designed based on 25 templates and include answers that incorporate the 1,000 most common nouns from the Brown Corpus, as well as 500 vocabulary items that are not present in the charts. The most recent dataset in our evaluation, ChartQA [9], includes two distinct subsets. ChartQA-H contains 4,800 charts and 9,600 questions penned by crowdsourced workers. ChartQA-M, on the other hand, features 17,100 charts and 23,100 questions. These questions are generated by a T5 model based on human-written summaries, allowing for a more open vocabulary and nuanced interpretation of charts.

Name	Data Types	Chart Types	Question Types	Answer Types
FigureQA	Synthetic	Synthetic bar/line/pie charts	Template-based	Yes/no
DVQA	Synthetic	Synthetic bar charts	Template-based	Fixed vocabulary
ChartQA	Real-world	Real-world bar/line/pie charts	Mixed	Open vocabulary

Table 1. Comparison between Existing CQA Datasets

For all the datasets under consideration, we employed the Adam optimizer with an initial learning rate of 5×10^{-4}. The chart question-answering component of our model was subjected to training over 20 epochs, utilizing a linear learning rate scheduler to adjust the learning rate dynamically. For a more rigorous comparative analysis between our approach and the previous state-of-the-art (SOTA) methods, we adopt the identical evaluation metric as outlined in ChartQA [9]. In this schema, a numerical answer is deemed correct if it falls within a 5% range of the gold standard answer. Conversely, for non-numeric responses, an exact match with the gold standard is requisite for the answer to be classified as correct.The empirical results of this configuration are summarized in Table 2.

Overall, our T5 model exhibited a slight advantage over the ChartQA version of T5 model when evaluated on FigureQA and DVQA datasets. However, the model's efficacy diminished when tested on the more intricate ChartQA dataset. Upon

Model	FigureQA		DVQA		ChartQA	
	Val 1	Val2	Test-Familiar	Test-Novel	Val	Test
FigureQA	72.54%	72.40%	-	-	-	-
DVQA	-	-	56.48%	56.62%	-	-
T5(ChartQA)	87.97%	87.83%	89.01%	76.89%	**40.15%**	**41.04%**
T5(Ours)	**91.50%**	**90.02%**	**91.75%**	**85.30%**	29.72%	27.55%

Table 2. Evaluation results

dissecting the sources of errors, we ascertained that the primary source of errors stems from the chart data extraction component.

A closer inspection of misclassified samples revealed that our model performed notably worse on line charts compared to bar and pie charts. This divergence in performance may be attributed to the inherent geometrical complexities associated with line segments in line charts, complicating the model's ability to generalize detection rules across different chart types. Another limitation surfaced in scenarios where the charts contained densely clustered components, causing the keypoint detection model to falter.

5. Conclusion

In this paper, we introduced ChartLLM, a novel pipeline aimed at enhancing the multimodal capabilities of large language models in processing chart images. By employing a two-stage approach comprising of key point detection using an hourglass network and table-based question answering via T5 models, we managed to convert intricate chart information into a more digestible table format. This in turn allows for more efficient and accurate chart question-answering capabilities for large language models.

We extensively evaluated ChartLLM on a range of datasets, both early and recent. Our evaluation showcased the model's capability to handle a diverse array of chart types and questions, thereby confirming its robustness and versatility. A distinguishing aspect of our approach lies in the uniformity of our chart information extraction mechanism. Unlike prior frameworks, such as ChartOCR [8], which necessitate distinct models for diverse chart types, our pipeline sidesteps this complexity. Leveraging an optimized key-point detection algorithm, we obviate the need for intricate, type-specific heuristics. Consequently, our single, unified model can handle a wide range of chart categories seamlessly.

However, there is room for future enhancements. One possible direction would be to extend the pipeline to more complex, multi-faceted chart types, or to integrate it with other types of visual data like maps and infographics. The other possible direction is to use the adapter method instead of fine-tuning the whole model, reducing computational cost for training.

In conclusion, ChartLLM represents a step forward in improving the ability of large language models on the interpret of charts and lays the groundwork for further exploration and improvements in the field of multimodal machine learning.

References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. 7

[2] Samira Ebrahimi Kahou, Adam Atkinson, Vincent Michalski, Akos Kadar, Adam Trischler, and Yoshua Bengio. Figureqa: An annotated figure dataset for visual reasoning. In *Visually grounded interaction and language workshop, NIPS 2017*, December 2017. 6, 7, 14

[3] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online, July 2020. Association for Computational Linguistics. 6

[4] Nengzheng Jin, Joanna Siebert, Dongfang Li, and Qingcai Chen. A survey on table question answering: Recent advances, 2022. 6

[5] Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. Dvqa: Understanding data visualizations via question answering. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5648–5656, 2018. 6, 7, 14

[6] Kushal Kafle, Robik Shrestha, Scott Cohen, Brian Price, and Christopher Kanan. Answering questions about data visualizations using efficient bimodal fusion. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020. 6

[7] Dae Hyun Kim, Vidya Setlur, and Maneesh Agrawala. Towards understanding how readers integrate charts and captions: A case study with line charts. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. 6

[8] Junyu Luo, Zekun Li, Jinpeng Wang, and Chin-Yew Lin. Chartocr: Data extraction from charts images via a deep hybrid framework. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1916–1924, 2021. 7, 15

[9] Ahmed Masry, Do Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2263–2279, Dublin, Ireland, May 2022. Association for Computational Linguistics. 6, 7, 14

[10] Nitesh Methani, Pritha Ganguly, Mitesh M. Khapra, and Pratyush Kumar. Plotqa: Reasoning over scientific plots. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1516–1525, 2020. 6, 7

[11] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation, 2016. 7

[12] Barlas Oguz, Xilun Chen, Vladimir Karpukhin, Stan Peshterliev, Dmytro Okhonko, Michael Schlichtkrull, Sonal Gupta, Yashar Mehdad, and Scott Yih. Unik-qa: Unified representations of structured and unstructured knowledge for open-domain question answering, 2022. 6

[13] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 10 2019. 6, 7, 14

[14] Hrituraj Singh and Sumit Shekhar. STL-CQA: Structure-based transformers with localization and encoding for chart question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3275–3284, Online, Nov. 2020. Association for Computational Linguistics. 7

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. 7

A. Featured Source Code

A.1. Model Backbone

```
class kp_module(nn.Module):
    def __init__(
        self, n, dims, modules, layer=residual,
        make_up_layer=make_layer, make_low_layer=make_layer,
        make_hg_layer=make_layer, make_hg_layer_revr=make_layer_revr,
        make_pool_layer=make_pool_layer, make_unpool_layer=make_unpool_layer,
        make_merge_layer=make_merge_layer, **kwargs
    ):
        super().__init__()

        self.n   = n

        curr_mod = modules[0]
        next_mod = modules[1]

        curr_dim = dims[0]
        next_dim = dims[1]

        self.up1  = make_up_layer(
            3, curr_dim, curr_dim, curr_mod,
            layer=layer, **kwargs
        )
        self.max1 = make_pool_layer(curr_dim)

        self.low1 = make_hg_layer(
            3, curr_dim, next_dim, curr_mod,
            layer=layer, **kwargs
        )
        self.low2 = kp_module(
            n - 1, dims[1:], modules[1:], layer=layer,
            make_up_layer=make_up_layer,
            make_low_layer=make_low_layer,
            make_hg_layer=make_hg_layer,
            make_hg_layer_revr=make_hg_layer_revr,
            make_pool_layer=make_pool_layer,
            make_unpool_layer=make_unpool_layer,
            make_merge_layer=make_merge_layer,
            **kwargs
        ) if self.n > 1 else \
        make_low_layer(
            3, next_dim, next_dim, next_mod,
            layer=layer, **kwargs
        )
        self.low3 = make_hg_layer_revr(
            3, next_dim, curr_dim, curr_mod,
            layer=layer, **kwargs
        )
        self.up2  = make_unpool_layer(curr_dim)

        self.merge = make_merge_layer(curr_dim)
```

```
    def forward(self, x):
        up1  = self.up1(x)
        max1 = self.max1(x)
        low1 = self.low1(max1)
        low2 = self.low2(low1)
        low3 = self.low3(low2)
        up2  = self.up2(low3)
        return self.merge(up1, up2)

class kp_group(nn.Module):
    def __init__(
        self, n, nstack, dims, modules, out_dim, pre=None, cnv_dim=256,
        make_cnv_layer=make_cnv_layer, make_heat_layer=make_kp_layer,
        make_regr_layer=make_kp_layer,
        make_up_layer=make_layer, make_low_layer=make_layer,
        make_hg_layer=make_layer, make_hg_layer_revr=make_layer_revr,
        make_pool_layer=make_pool_layer, make_unpool_layer=make_unpool_layer,
        make_merge_layer=make_merge_layer, make_inter_layer=make_inter_layer,
        kp_layer=residual
    ):
        super(kp_group, self).__init__()
        self.nstack    = nstack
        self._decode   = _decode_group
        curr_dim = dims[0]
        self.pre = nn.Sequential(
            convolution(7, 3, 128, stride=2),
            residual(3, 128, 256, stride=2)
        ) if pre is None else pre
        self.kps  = nn.ModuleList([
            kp_module(
                n, dims, modules, layer=kp_layer,
                make_up_layer=make_up_layer,
                make_low_layer=make_low_layer,
                make_hg_layer=make_hg_layer,
                make_hg_layer_revr=make_hg_layer_revr,
                make_pool_layer=make_pool_layer,
                make_unpool_layer=make_unpool_layer,
                make_merge_layer=make_merge_layer
            ) for _ in range(nstack)
        ])
        self.cnvs = nn.ModuleList([
            make_cnv_layer(curr_dim, cnv_dim) for _ in range(nstack)
        ])

        self.key_cnvs = nn.ModuleList([
            make_cnv_layer(cnv_dim, cnv_dim) for _ in range(nstack)
        ])
        self.center_cnvs = nn.ModuleList([
            make_cnv_layer(cnv_dim, cnv_dim) for _ in range(nstack)
        ])

        self.key_heats = nn.ModuleList([
```

```
            make_heat_layer(cnv_dim, curr_dim, out_dim) for _ in range(nstack)
        ])
        self.center_heats = nn.ModuleList([
            make_heat_layer(cnv_dim, curr_dim, out_dim) for _ in range(nstack)
        ])

        for key_heat, center_heat in zip(self.key_heats, self.center_heats):
            key_heat[-1].bias.data.fill_(-2.19)
            center_heat[-1].bias.data.fill_(-2.19)

        self.inters = nn.ModuleList([
            make_inter_layer(curr_dim) for _ in range(nstack - 1)
        ])

        self.inters_ = nn.ModuleList([
            nn.Sequential(
                nn.Conv2d(curr_dim, curr_dim, (1, 1), bias=False),
                nn.BatchNorm2d(curr_dim)
            ) for _ in range(nstack - 1)
        ])
        self.cnvs_   = nn.ModuleList([
            nn.Sequential(
                nn.Conv2d(cnv_dim, curr_dim, (1, 1), bias=False),
                nn.BatchNorm2d(curr_dim)
            ) for _ in range(nstack - 1)
        ])

        self.key_regrs = nn.ModuleList([
            make_regr_layer(cnv_dim, curr_dim, 2) for _ in range(nstack)
        ])
        self.center_regrs = nn.ModuleList([
            make_regr_layer(cnv_dim, curr_dim, 2) for _ in range(nstack)
        ])

        self.relu = nn.ReLU(inplace=True)

        encoder_layer = nn.TransformerEncoderLayer(d_model=260, nhead=4,
            dim_feedforward = 1024)
        self.transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers
            =6)

        self.classifier = nn.Sequential(
            nn.Linear(260, 64),
            nn.ReLU(inplace=True),
            nn.Linear(64, 2)
        )


    def _train(self, *xs):

        image       = xs[0]
        key_inds    = xs[1]
        center_inds = xs[2]
```

```
key_lens    = xs[3]
center_lens = xs[4]
inter = self.pre(image)
outs  = []
layers = zip(
    self.kps, self.cnvs,
    self.key_cnvs, self.center_cnvs,
    self.key_heats, self.center_heats,
    self.key_regrs, self.center_regrs
)
for ind, layer in enumerate(layers):
    kp_, cnv_          = layer[0:2]
    key_cnv_, center_cnv_   = layer[2:4]
    key_heat_, center_heat_ = layer[4:6]
    key_regr_, center_regr_ = layer[6:8]
    kp  = kp_(inter)
    cnv = cnv_(kp)
    key_cnv = key_cnv_(cnv)
    center_cnv = center_cnv_(cnv)
    key_heat, center_heat = key_heat_(key_cnv), center_heat_(center_cnv)
    key_regr, center_regr = key_regr_(key_cnv), center_regr_(center_cnv)
    #print(f"Shape of key_regrs {key_regrs.shape}")
    key_regr = _transpose_and_gather_feat(key_regr, key_inds)
    center_regr = _transpose_and_gather_feat(center_regr, center_inds)
    outs += [key_heat, center_heat, key_regr, center_regr]
    if ind < self.nstack - 1:
        inter = self.inters_[ind](inter) + self.cnvs_[ind](cnv)
        inter = self.relu(inter)
        inter = self.inters[ind](inter)

batch_size = key_cnv.shape[0]
_, _, height, width = key_cnv.size()
key_feat = _transpose_and_gather_feat(key_cnv, key_inds)
center_feat = _transpose_and_gather_feat(center_cnv, center_inds)

group_preds = []
for b_ind in range(batch_size):
    if center_lens[b_ind] == 0 or key_lens[b_ind] == 0: continue
    cen_len = min(center_lens[b_ind], 40)
    cen_emb = center_feat[b_ind][:cen_len, :]
    tmp_inds = center_inds[b_ind][:cen_len].float()
    cen_pos = torch.stack([(tmp_inds % width) / width , (tmp_inds // width
        ) / height]).transpose(0,1)
    cen_type = torch.ones((cen_pos.size(0),1)).float().cuda()
    cen_emb = torch.cat((cen_emb, cen_pos, cen_type), 1)
    cen_type2 = torch.eye(cen_emb.size(0)).unsqueeze(-1).cuda()
    cen_emb = cen_emb.unsqueeze(1).repeat(1,cen_emb.size(0),1)
    cen_emb = torch.cat((cen_emb, cen_type2), -1)
    key_emb = key_feat[b_ind][:key_lens[b_ind], :]
    tmp_inds = key_inds[b_ind][:key_lens[b_ind]].float()
    key_pos = torch.stack([(tmp_inds % width) / width , (tmp_inds // width
        ) / height]).transpose(0,1)
    key_type = torch.zeros((key_pos.size(0),2)).float().cuda()
```

```
            key_emb = torch.cat((key_emb, key_pos, key_type), 1)
            key_emb = key_emb.unsqueeze(1).repeat(1, cen_len, 1) # key_len *
                cen_len (batch_len) * featdim
            src = torch.cat((cen_emb, key_emb), 0)
            out = self.transformer_encoder(src).transpose(1,0)
            out = self.classifier(out)
            group_preds.append(out.reshape(-1,2))

        return (outs, tuple(group_preds))

    def _test(self, *xs, **kwargs):
        image = xs[0]

        inter = self.pre(image)
        outs  = []

        layers = zip(
            self.kps, self.cnvs,
            self.key_cnvs, self.center_cnvs,
            self.key_heats, self.center_heats,
            self.key_regrs, self.center_regrs
        )
        for ind, layer in enumerate(layers):
            kp_, cnv_          = layer[0:2]
            key_cnv_, center_cnv_   = layer[2:4]
            key_heat_, center_heat_ = layer[4:6]
            key_regr_, center_regr_ = layer[6:8]

            kp  = kp_(inter)
            cnv = cnv_(kp)

            if ind == self.nstack - 1:
                key_cnv = key_cnv_(cnv)
                center_cnv = center_cnv_(cnv)

                key_heat, center_heat = key_heat_(key_cnv), center_heat_(
                    center_cnv)
                key_regr, center_regr = key_regr_(key_cnv), center_regr_(
                    center_cnv)

                outs += [key_heat, center_heat, key_regr, center_regr]

            if ind < self.nstack - 1:
                inter = self.inters_[ind](inter) + self.cnvs_[ind](cnv)
                inter = self.relu(inter)
                inter = self.inters[ind](inter)

        detections_key, detections_cen, key_inds, center_inds, key_scores,
            center_scores = self._decode(*outs[-4:], **kwargs)

        b_ind = 0
        _, _, height, width = key_cnv.size()
        key_feat    = _transpose_and_gather_feat(key_cnv, key_inds)
```

```
        center_feat = _transpose_and_gather_feat(center_cnv, center_inds)
        key_len = (key_scores[b_ind] > 0.4).sum()
        cen_len = (center_scores[b_ind] > 0.4).sum()
        if key_len == 0 or cen_len == 0: return detections_key, detections_cen,
            torch.zeros((1,1))

        cen_emb = center_feat[b_ind][:cen_len, :]
        tmp_inds= center_inds[b_ind][:cen_len].float()
        cen_pos = torch.stack([(tmp_inds % width) / width , (tmp_inds // width) /
            height]).transpose(0,1)
        cen_type = torch.ones((cen_pos.size(0),1)).float().cuda()
        cen_emb = torch.cat((cen_emb, cen_pos, cen_type), 1)
        cen_type2 = torch.eye(cen_emb.size(0)).unsqueeze(-1).cuda()
        cen_emb = cen_emb.unsqueeze(1).repeat(1,cen_emb.size(0),1)
        cen_emb = torch.cat((cen_emb, cen_type2), -1)
        key_emb = key_feat[b_ind][:key_len, :]
        tmp_inds = key_inds[b_ind][:key_len].float()
        key_pos = torch.stack([(tmp_inds % width) / width , (tmp_inds // width) /
            height]).transpose(0,1)
        key_type = torch.zeros((key_pos.size(0),2)).float().cuda()
        key_pos = torch.cat((key_pos, key_type), 1)
        key_emb = torch.cat((key_emb, key_pos), 1)
        key_emb = key_emb.unsqueeze(1).repeat(1, cen_emb.size(1), 1)

        src = torch.cat((cen_emb, key_emb), 0)
        out = self.transformer_encoder(src).transpose(1,0)
        out = self.classifier(out)
        out = nn.functional.softmax(out, dim = -1)
        out = out[:,:, 1]

        return detections_key, detections_cen, out

    def forward(self, *xs, **kwargs):
        if len(xs) > 1:
            return self._train(*xs, **kwargs)
        return self._test(*xs, **kwargs)

class GroupingLoss(nn.Module):
    def __init__(self, lambda_, lambda_b, regr_weight=1, focal_loss=_neg_loss):
        super(GroupingLoss, self).__init__()

        self.regr_weight = regr_weight
        self.focal_loss  = focal_loss
        self.regr_loss   = _regr_loss
        self.lambda_ = lambda_
        self.lambda_b = lambda_b
        self.group_loss = nn.CrossEntropyLoss()
    def forward(self, outputs, targets):
        stride = 4

        outs, group_preds = outputs

        key_heats = outs[0::stride]
```

```
        center_heats = outs[1::stride]
        key_regrs = outs[2::stride]
        center_regrs = outs[3::stride]
        gt_key_heat = targets[0]
        gt_center_heat = targets[1]
        gt_key_mask    = targets[2]
        gt_center_mask    = targets[3]
        gt_key_regr = targets[4]
        gt_center_regr = targets[5]
        group_targets = targets[6]
        tag_lens_center = targets[7]
        tag_lens_key = targets[8]
        focal_loss = 0
        key_heats = [_sigmoid(t) for t in key_heats]
        center_heats = [_sigmoid(b) for b in center_heats]

        focal_loss += self.focal_loss(key_heats, gt_key_heat, self.lambda_, self.
            lambda_b) / 2
        focal_loss += self.focal_loss(center_heats, gt_center_heat, self.lambda_,
            self.lambda_b)
        regr_loss = 0
        for key_regr, center_regr in zip(key_regrs, center_regrs):
            regr_loss += self.regr_loss(key_regr, gt_key_regr, gt_key_mask)
            regr_loss += self.regr_loss(center_regr, gt_center_regr,
                gt_center_mask)
        regr_loss = self.regr_weight * regr_loss

        group_targets_trim = []
        for b_ind in range(group_targets.size(0)):
            cen_len = min(tag_lens_center[b_ind], 40)
            tmp = group_targets[b_ind][:cen_len, :tag_lens_key[b_ind]]
            tmp = torch.cat((torch.zeros((cen_len,cen_len)).cuda().long(), tmp),
                1)
            if tmp.reshape(-1).size(0) == 0: continue
            group_targets_trim.append(tmp.reshape(-1))

        group_loss = 0
        for b_ind in range(len(group_targets_trim)):
            group_loss += self.group_loss(group_preds[b_ind], group_targets_trim[
                b_ind])
        group_loss = 10 * group_loss # lr = 0.000025

        if group_loss == 0:
            loss = (focal_loss + regr_loss) / len(key_heats)
        else:
            loss = (focal_loss + regr_loss + group_loss) / len(key_heats)
        return loss.unsqueeze(0)

from .py_utils import kp_group, GroupingLoss, _neg_loss, residual
from .model_utils import make_pool_layer, make_hg_layer

class Model(kp_group):
    def __init__(self, db):
```

```
        n = 5
        dims = [256, 256, 384, 384, 384, 512]
        modules = [2, 2, 2, 2, 2, 4]
        out_dim = 10

        super(Model, self).__init__(
            n, 2, dims, modules, out_dim,
            make_pool_layer=make_pool_layer,
            make_hg_layer=make_hg_layer,
            kp_layer=residual, cnv_dim=256
        )


loss = GroupingLoss(focal_loss=_neg_loss, lambda_=4, lambda_b=2)
```

A.2. Implementation Details

Table 3. Custom Category Labels for Chart Components

Label Number	Chart Component
1	Bars in bar charts
2	Lines in line charts
3	Pies in pie charts
4	Legends
5	Title of the values axes
6	Title of the entire chart
7	Title of the category axes

Listing 1. Output table of the example bar image

```
Aakna + da aebe + /wla | Huiri/a+als/ | D | I | E | A | EO | +KA &
34321 | 33192 | 32473 | 23898 | 15099 | 12003 | 9873 | 8897
Chart Type: v_bar_categorical
Title: None
x_axis_title: None
y_Axis_Title: aDI
```

B. Acknowledgement

In December 2022, every single day brought a deluge of news reports covering various topics, flooding my social media feeds. These reports often included charts illustrating important data, such as the weekly COVID-19 infection rates, annual CPI trends in the U.S., or price fluctuations for specific products over five-year spans. I sought to find a tool that could summarize these news reports effectively. To my surprise, I discovered that while existing models like ChatGPT, T5, and BART were proficient at answering text-based questions, they were not equipped to handle chart images. This glaring gap in existing technology drove me to delve deeper into this subject matter.

In the journey of completing this paper, I am first deeply indebted to Dr. Zhiqi Cheng from Carnegie Mellon University, whose unparalleled expertise and keen insights were instrumental in shaping this research. His constructive critiques and continuous encouragement fostered both a robust research methodology and my own intellectual growth. Dr. Cheng's dedication to teaching goes beyond mere academic guidance; it serves as a lifelong lesson in perseverance and excellence. I am truly grateful for the invaluable mentorship he has provided.

I owe immense gratitude to my parents, who have ceaselessly nurtured my interests in information technology and artificial intelligence. Their encouragement and belief in me have played an instrumental role in my quest to become a computer scientist.

I'd also like to acknowledge the Center for High Performance Computing at Shanghai Jiao Tong University, which provided the computational resources for this paper. The calculations were performed on the π 2.0 (or Siyuan-1) cluster, which greatly aided the progress of this research.

Lastly, I would like to express my heartfelt thanks to the S.T. Yau Science Award for offering me an incredible opportunity and platform to showcase my research. This experience has been integral in my academic development and has provided me with the visibility to engage with a broader scientific community.