

# TinyChart: Efficient Chart Understanding with Visual Token Merging and Program-of-Thoughts Learning

Liang Zhang<sup>\*†</sup>

Renmin University of China  
Beijing, China  
zhangliang00@ruc.edu.cn

Anwen Hu\*

Alibaba Group  
Beijing, China  
huanwen.haw@alibaba-inc.com

Haiyang Xu

Alibaba Group  
Hangzhou, China  
shuofeng.xhy@alibaba-inc.com

Ming Yan

Alibaba Group  
Hangzhou, China  
ym119608@alibaba-inc.com

Yichen Xu

Renmin University of China  
Beijing, China  
xu\_yichen@ruc.edu.cn

Qin Jin<sup>‡</sup>

Renmin University of China  
Beijing, China  
qjin@ruc.edu.cn

Ji Zhang

Alibaba Group  
Hangzhou, China  
zj122146@alibaba-inc.com

Fei Huang

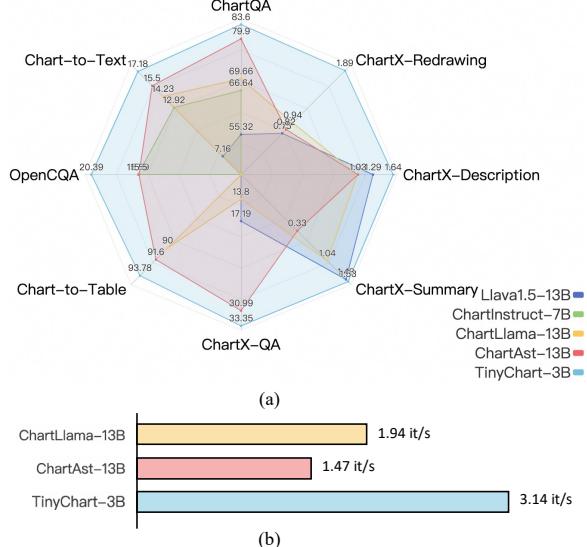
Alibaba Group  
Hangzhou, China  
f.huang@alibaba-inc.com

## ABSTRACT

Charts are important for presenting and explaining complex data relationships. Recently, multimodal large language models (MLLMs) have shown remarkable capabilities in various chart understanding tasks. However, the sheer size of these models in terms of parameters and computational requirements limits their use in resource-constrained environments. In this paper, we present TinyChart, an efficient MLLM for chart understanding with only 3B parameters. TinyChart overcomes two key challenges in efficient chart understanding: (1) reduce the burden of learning numerical computations through a Program-of-Thoughts (PoT) learning strategy, which trains the model to generate Python programs for numerical calculations, and (2) reduce lengthy vision feature sequences produced by the vision transformer for high-resolution images through a Vision Token Merging module, which gradually merges most similar vision tokens. Extensive experiments demonstrate that our 3B TinyChart achieves SOTA performance on a variety of chart understanding benchmarks including ChartQA, Chart-to-Text, Chart-to-Table, OpenCQA, and ChartX. It outperforms several chart understanding MLLM with up to 13B parameters such as ChartLlama and ChartAst, and close-sourced general-purpose MLLM GPT-4V on ChartQA. It also demonstrates its superior efficiency with higher throughput during inference due to a smaller model scale and more efficient vision encoding. Our code and model are available at <https://github.com/X-PLUG/mPLUG-DocOwl/tree/main/TinyChart>.

## 1 INTRODUCTION

As an important information source, charts can intuitively visualize data in various visual presentation forms and have become an indispensable part of information dissemination, business decision-making, and academic research [16]. With the rapid growth of multimodal data, automatically comprehending charts has become



**Figure 1: Our TinyChart-3B outperforms several 13B MLLMs on a variety of chart understanding benchmarks (a), while achieving larger inference throughput (b).**

a pressing need and received increasing attention from the research community [3, 10, 34, 35]. Recently, Multimodal Large Language Models (MLLMs) have shown strong capability in comprehending images and following instructions [5, 6, 25, 29, 30, 39, 53, 55, 59]. Based on these MLLMs, some recent works [10, 14, 34, 35] further build chart understanding models by collecting and constructing versatile chart comprehension datasets and performing supervised fine-tuning.

However, despite their remarkable success, current chart understanding models still face three main limitations: (1) Considerable amount of parameters makes training and deployment challenging. For example, ChartLlama [10] is a model with 13 billion parameters, which is hard to deploy on a single consumer GPU with less than

<sup>\*</sup>Authors contributed equally.

<sup>†</sup>Work done during an internship at Alibaba Group.

<sup>‡</sup>Corresponding author.

26GB of VRAMs. (2) They are prone to errors when tackling questions involving numerical calculations [35], which are difficult to directly answer without any reasoning steps. (3) They struggle with efficiently encoding for high-resolution images since the standard vision transformer would produce lengthy feature sequences.

To overcome such limitations in chart understanding, we propose an efficient and powerful MLLM, namely **TinyChart**. As shown in Figure 1, through the efficient visual encoding and Program-of-Thoughts learning strategy, TinyChart achieves state-of-the-art performances on various chart understanding benchmarks with only 3B parameters, while excelling in faster inference throughput.

For efficient visual encoding, we propose to merge visual tokens based on the observation that chart images often contain large areas of color and white spaces. Inspired by [2], we adopt a parameter-free Visual Token Merging module inside each vision transformer layer, which aggregates the most similar visual tokens and gradually reduces the length of the visual feature sequence, thus making it possible to efficiently encode high-resolution chart images. This enables the model to maintain high-resolution chart image input while controlling the computation load.

Moreover, inspired by [4], we propose Program-of-Thoughts learning that enhances the model’s ability to resolve mathematical problems. According to statistics on ChartQA [32], 42% of questions for charts require numerical answers, and most existing models struggle to perform numerical question answering [27, 35]. To learn chart understanding more efficiently, we train the model to generate Python programs for the computation problems step by step. The programs are then passed to a Python interpreter to produce the final answer. To support Program-of-Thoughts learning, we further construct the ChartQA-PoT dataset based on ChartQA [32]. The QA pairs in our ChartQA-PoT are constructed in two ways: (1) Template-based PoT construction, which generates questions and programs by filling in the manually written templates based on chart data. (2) GPT-based PoT construction, which leverages gpt-3.5-turbo [38] to generate programs based on human-written questions. Experimental results show that Program-of-Thoughts learning can significantly improve the question-answering, especially numerical question answering ability of TinyChart.

The main contributions of this work are as follows:

- We introduce TinyChart, an efficient multimodal chart understanding model, which outperforms several 13B MLLMs and achieves state-of-the-art performances on a variety of chart understanding benchmarks, while excelling in faster inference speed at the same time.
- We propose a Program-of-Thoughts (PoT) learning strategy to enhance the model in learning numerical calculation and carefully build a PoT dataset ChartQA-PoT.
- We adopt Visual Token Merging for efficient vision encoding, which significantly reduces the length of vision feature sequences and enables the model to encode high-resolution chart images with constrained computing resources.

## 2 RELATED WORK

### 2.1 Chart Understanding

Chart understanding requires the model to comprehend chart contents and accomplish related tasks specified by the instructions.

This field encompasses low-level recognition tasks, such as data extraction [26], and high-level tasks, such as question-answering (QA) [19, 32, 36], summarization [21, 37], and re-rendering [10]. As charts often contain OCR text pivotal for data interpretation, and many instructions require the model to perform numerical calculations, chart understanding demands robust text recognition capabilities and computational reasoning from the model. Early approaches [9, 13, 26, 36, 44, 58] rely on pipeline methods that use off-the-shelf OCR tools or component detectors to transform charts into data tables and other textual representations. They then employ language models to complete the specified tasks. These pipeline approaches, limited by their inability to optimize jointly, were hampered by error accumulation. Recent studies [10, 27, 28, 33–35] have shifted towards end-to-end methods based on multimodal large language models. These studies adopt the structure of multimodal large language models [25, 29, 30, 53, 55] and enhance chart understanding abilities through supervised fine-tuning [40] with substantial chart instruction data [10, 34, 35]. Although these models demonstrate improvement in performance, their extensive parameter size prevents them from being easily trained or deployed under resource-constrained scenarios. In this paper, we demonstrate that a 3B MLLM is enough to achieve state-of-the-art performance on several chart understanding tasks. Meanwhile, it has been well observed that these models are prone to numerical errors [27, 34, 35]. Though Meng et al. [35] try to construct executable command lines in JSON format based on a template to eliminate numerical errors, we argue that it is insufficient to fully address this issue for two reasons: 1) The executable command lines in JSON format produced by Meng et al. [35] relies on a specific computational backend, which limits their potential versatility. 2) Template-based programs can only cover rather limited scenarios. Instead, we construct the Program-of-Thoughts learning dataset with the combination of both templates and GPT-generated programs. This allows the model to more effectively learn how to solve numerical problems.

### 2.2 Multimodal Large Language Model

Multimodal large language models (MLLM) exhibit strong capabilities in visual understanding and instruction following [39, 46]. They typically comprise transformer-based visual encoders, large language models, and vision-language connectors [5, 29, 30, 53–55, 59, 61]. These models are generally trained on extensive general image-text data for cross-modal alignment and instruction fine-tuning. Although some studies have showcased a degree of OCR capability in these multimodal large language models [31, 60], their performance on document and chart understanding benchmarks remains suboptimal due to their low input resolution [6, 52]. Efforts in the general document domain have attempted to improve the fine-grained understanding capabilities of MLLMs by increasing resolution [1], segmenting images [6, 15, 25, 52], utilizing frequency domain signals [8], and introducing additional high-resolution encoders [12]. However, these models often suffer from low efficiency, primarily due to the excessive length of the high-resolution visual sequences. The visual token merging method adopted in this paper can significantly reduce the length of visual feature sequences and relax the computational requirements with high-resolution input.

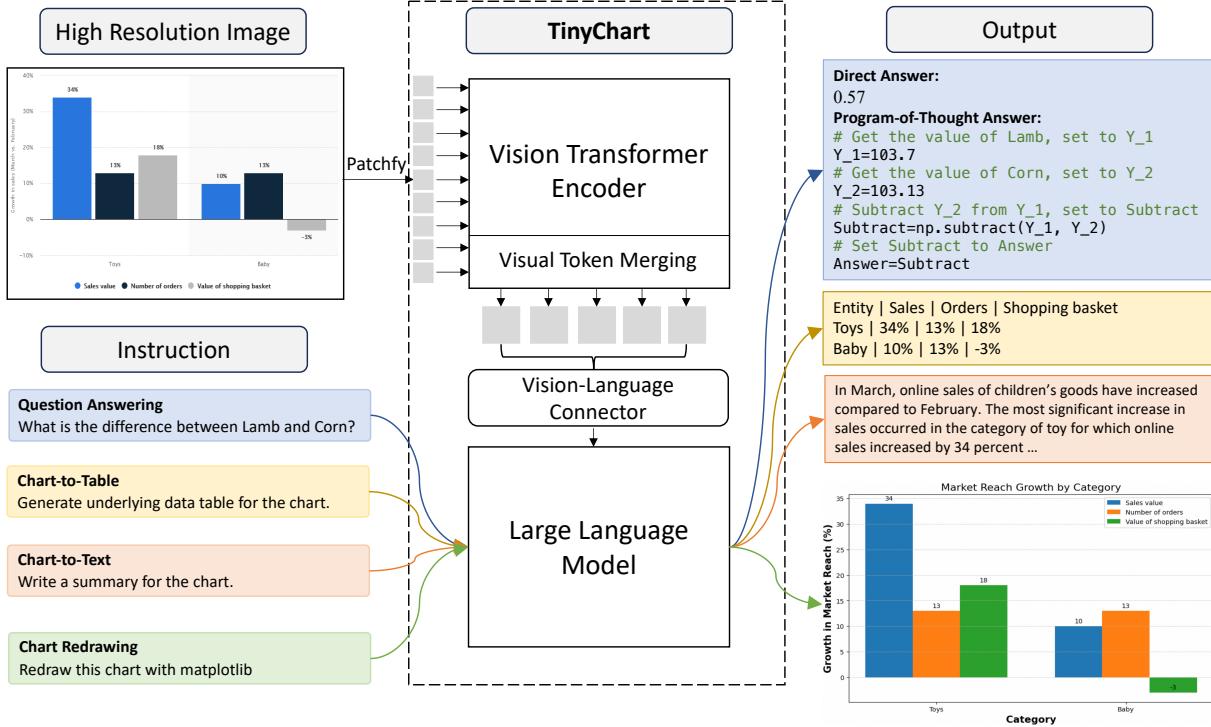


Figure 2: Overview of TinyChart.

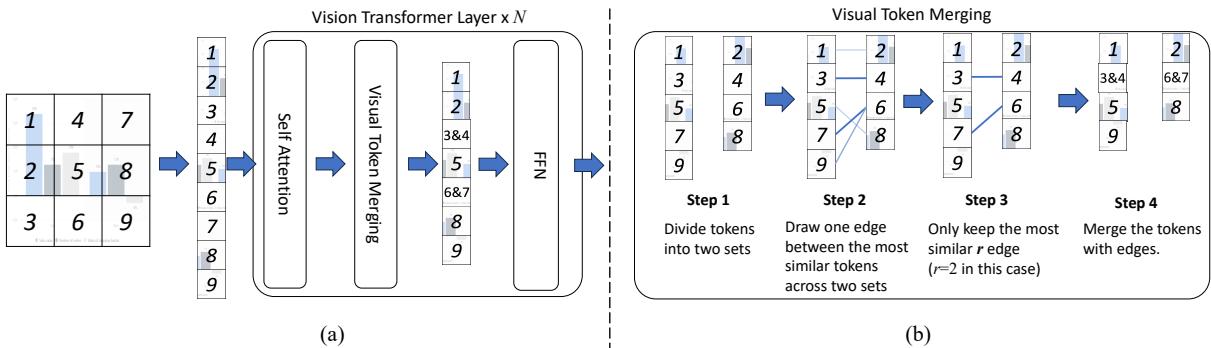


Figure 3: (a) Vision transformer layer with Visual Token Merging. (b) Process of the Visual Token Merging.

## 3 TINYCHART

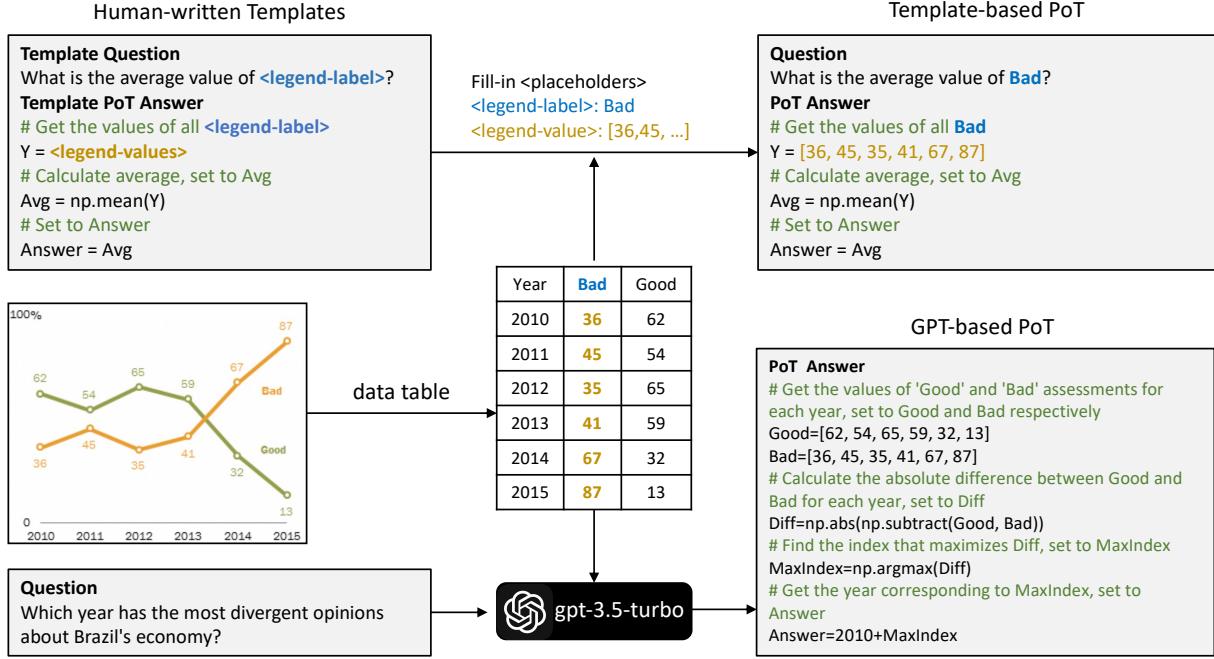
### 3.1 Model Architecture

Figure 2 shows the overview framework of our proposed TinyChart. It follows the typical architecture of the multimodal large language model (MLLM), which consists of a vision transformer encoder, a vision-language connector, and a large language model. To encode high-resolution visual input effectively, we insert the visual token merging module inside each vision transformer layer.

**3.1.1 Vision Transformer Encoder.** The vision transformer encoder aims to encode chart images into vision features. A standard vision transformer [7] first resizes the input image  $I$  into a fixed resolution and crops the image into patches. Then the patches are treated as vision tokens and processed with transformer encoder layers [48].

Suppose the input image  $I^{N \times N}$  is in resolution  $N \times N$ , and the patch size is  $P \times P$ , the length of vision tokens would be  $(N/P)^2$ . Since the standard transformer layer does not reduce the sequence length, the vision transformer finally produces a vision feature in length  $(N/P)^2$ . In practice, when  $N$  is large, the vision feature can be very long and inefficient for the language model to handle.

**Visual Token Merging** Since key information (such as OCR words) in a chart can be unrecognizable in low-resolution images [15], high-resolution input is essential for chart understanding. However, charts typically contain a large number of color blocks and blank spaces, where patches are visually similar. To achieve efficient and effective chart understanding, we apply Visual Token Merging [2] in each transformer layer. The process of Visual Token Merging is shown in Figure 3. By merging the  $r$  most similar token



**Figure 4: The demonstration of constructing Template-based PoT (upper half) and GPT-based PoT (lower half) in the ChartQA-PoT dataset.**

pairs, it reduces the length of the vision feature by  $r$  in each layer. We measure the similarity between two tokens using the cosine distance between Keys from self-attention following [2]. As shown in the lower part of Figure 3, Vision Token Merger finds the top- $r$  similar token pairs through bipartite graph matching. It first divides the vision tokens into two disjoint sets. Then, for each token in one set, it finds the most similar tokens in the other set and draws an edge between the two tokens. After that, it only keeps the top- $r$  most similar edges and merges the features of the two endpoints through average pooling. Note that not only spatially adjacent visual tokens are subject to merging. Non-adjacent tokens can also be merged if they belong to different subsets and are similar enough.

**Proportional attention** The visual token merging operation aggregates tokens with a similar feature into one. Therefore, it will reduce the proportion of this visual feature in the attention calculation in the following transformer layer, since the number of this feature has decreased. To solve this issue, we let the attention operation consider the actual number of patches  $s$  represented by each token as follows:

$$\text{Attention} = \text{softmax} \left( \frac{QK^\top}{\sqrt{d}} + \log s \right) V \quad (1)$$

Where  $Q, K, V$  denotes the query, key, and value of self-attention which are linear projected from the hidden states [48]. By adding  $\log s$  inside softmax, the token that merged from  $s$  patches are duplicated by  $s$  times in the attention calculation [2].

**3.1.2 Vision-Language Connector.** The vision language connector aims to project the vision features into the embedding space of the large language model. Following [29, 61], we implement the

vision-language connector as a multiple-layer perceptron with one hidden layer and GeLU [11] activation.

**3.1.3 Large Language Model.** The large language model aims to comprehend both visual features and language instructions, and then generate responses to accomplish chart understanding tasks. It is implemented as a transformer decoder [48] with a causal attention mask. The training objective of the model is language modeling. Assuming the visual features is  $V$ , the language instruction is  $L$ , and the response is  $R$ , then the loss function is defined as follows:

$$\mathcal{L} = \frac{1}{T} \sum_{i=1}^T \text{LLM}(R_i | V, L, R_{<i}) \quad (2)$$

Where  $T$  is the number of tokens in  $R$ . Note that we only calculate loss over tokens in the responses following the supervised fine-tuning setting in [29].

### 3.2 Program-of-Thoughts Learning

Program-of-Thoughts (PoT) learning aims to enhance the learning efficiency of models for numerical computation. In PoT learning, the model is trained to generate executable Python codes as the target of a question. The final answer is obtained by executing the code with a Python interpreter. Compared to short answers that only contain the calculated values, the Python code includes natural language comments and multi-step reasoning processes, offering a form of learning closely aligned with the pre-training of the large language model.

**ChartQA-PoT Dataset** To support PoT learning on chart understanding, we construct the ChartQA-PoT dataset based on the training split of ChartQA [32]. ChartQA-PoT contains 140,584 (question, PoT answer) pairs. Each PoT answer consists of multiple lines of Python code. We provide natural language comments for almost all code lines to explain their behaviors. We employ two approaches for constructing (question, PoT answer) pairs: Template-based PoT, and GPT-based PoT.

**3.2.1 Template-based PoT.** Based on the chart images in ChartQA, we construct template-based (question, PoT answer) pairs. As illustrated in the upper half of Figure 4, the Template-based PoT is constructed based on human-written templates containing placeholders for both questions and code. The template questions involve common numerical operations such as calculating the sum, average, minimal, and maximum values. We adopt the 40 template questions proposed by PlotQA [36] and manually write their corresponding template Python code to solve them. As shown in the top-left part of Figure 4, the template code consists of several variable assignment operations with NumPy [47] functions to perform calculations. The beginning steps usually involve extracting the relevant data from the chart and assigning them to variables. The final computed result is stored in a variable named "Answer". For each placeholder in the template, we identify all possible values from the data table of the chart and randomly select one to fill in the placeholder. After removing incorrect or unreasonable filled-ins using rule-based methods, we finally successfully construct 119,281 (question, PoT pairs) over 17,498 images from ChartQA.

**3.2.2 GPT-based PoT.** Although the template-based method allows for the construction of a large number of question-answer pairs, the diversity of these pairs is limited due to the fixed templates. To improve the generalization ability of PoT learning, we have additionally built GPT-generated PoT data by leveraging the powerful command-following and code-generation capabilities of large language models. Specifically, we prompt gpt-3.5-turbo [38] to generate PoT answers similar to the template PoT format for questions annotated in ChartQA using in-context examples. As shown in Figure 4, since gpt-3.5-turbo does not accept image input, we also provide the data table corresponding to the chart as text input to gpt-3.5-turbo. We screen the quality of the generated PoT answers by running them through a Python interpreter. If the annotated PoT answer can not run on the Python interpreter, or if the answer obtained is different from the annotated one in ChartQA, then the corresponding PoT Answer is deleted. In the end, we construct 21,303 (question, PoT Answer) pairs on 15,521 chart images.

### 3.3 Multitask Learning

We perform multitask learning to train our TinyChart model. We collect a chart understanding dataset that contains 1.36M samples for supervised fine-tuning. It covers various chart understanding tasks including chart question answering, chart-to-text generation, chart-to-table generation, and chart instruction following. Table 1 shows the collection of our training dataset. We mix data in different tasks together to jointly train the model, and use task-specified instructions to enable the model to differentiate between them.

**Table 1: Datasets used for training TinyChart.** The benchmark datasets consist of basic chart understanding evaluations including QA, summary, and chart-to-table generation. Note that in ablation studies, we only use the benchmark datasets for training due to limited computational resources.

Dataset	Benchmark	Samples
<b>Chart question answer</b>		
ChartQA [32]	✓	28,299
ChartQA-PoT	✓	140,584
PlotQA [36]		157,070
DVQA [19]		200,000
OpenCQA [20]		5,407
<b>Chart-to-text generation</b>		
Pew [21]	✓	7,892
Statista [21]	✓	29,589
OpenCQA [20]		5,407
Vistext [45]		11,171
ChartSumm [42]		75,255
Chart2Text-8k [37]		7,862
<b>Chart-to-table generation</b>		
ChartQA [32]	✓	19,373
PlotQA [36]		190,720
Chart2Text-8k		8,305
DVQA [19]		300,000
Statista [21]		29,589
<b>Chart instruction following</b>		
ChartLlama [10]		148,398
<b>Total</b>		<b>1,364,921</b>

The training objective is language modeling on response tokens as presented in Eq.2. Note that in ablation studies, we train solely with benchmark datasets due to limited computational resources.

## 4 EXPERIMENT

### 4.1 Implementation Details

TinyChart is initialized from TinyLlava [61], which utilizes the SigLIP [57] as the vision encoder and Phi-2 [23] as the large language model. The origin input resolution of the vision encoder is 384×384. We extend the input resolution to 512×512 and 768×768 and apply visual token merging with  $r = 20$  and  $r = 84$  in each transformer layer respectively. We train the entire model for 3 epochs with a batch size of 512. The learning rate is set to  $1e - 4$ , with a warmup in the beginning 3% steps, and then decays to 0 at the end of training. The total training process costs 3 days on 32 Tesla V100 GPUs with 32 GB VRAMs.

### 4.2 Evaluation Benchmarks

**ChartQA** ChartQA [32] aims to generate a short answer to the question based on the chart content. It includes a lot of questions that require numerical calculation. We report the relaxed accuracy that allows numerical error within 5% as the metric following [10, 32,

**Table 2: Main results on chart-related benchmarks. The inference throughput is evaluated on the ChartQA test with a batch size of 1 on V100 32GB.**

Model	#Parameters	Resolution	Inference Throughput	ChartQA			Chart-to-Text	Chart-to-Table	OpenCQA
				Aug.	Hum.	Avg.	BLEU4	RMS <sub>F1</sub>	BLEU4
<b><i>Close source models</i></b>									
GPT-4V [39]	-	-	-	-	-	78.50	-	-	-
Gemini-Ultra [46]	-	-	-	-	-	80.80	-	-	-
Qwen-VL-Max [1]	-	-	-	-	-	79.80	-	-	-
Deplet+Codex [26]	1.3B+175B	-	-	91.00	67.60	79.30	-	87.22	-
<b><i>Open source models</i></b>									
Llava1.5 [29]	13B	336×336	1.94 it/s	72.96	37.68	55.32	7.16	48.95	-
Qwen-VL [1]	9.6B	448×448	1.65 it/s	78.90	44.30	61.60	-	-	-
UReader [52]	7B	224×224(×20)	1.67 it/s	79.42	39.12	59.30	-	-	-
DocOwl1.5 [15]	8B	448×448(×9)	1.56 it/s	91.38	49.62	70.50	-	-	-
ChartInstruct [34]	7B	-	-	87.76	45.52	66.64	13.83	-	15.59
ChartLlama [10]	13B	336×336	1.94 it/s	90.36	48.96	69.66	14.23	90.00	-
ChartAst [35]	13B	448×448	1.47 it/s	<b>93.90</b>	65.90	79.90	15.50	91.60	15.50
TinyChart@512	3B	512×512	<b>3.65</b> it/s	93.60	72.16	82.88	<b>17.93</b>	92.93	19.62
TinyChart@768	3B	768×768	3.14 it/s	93.86	<b>73.34</b>	<b>83.60</b>	17.18	<b>93.78</b>	<b>20.39</b>

35]. Note that our TinyChart with Program-of-Thoughts learning can perform ChartQA in the following four settings:

- **Direct**: the model produces short answers directly.
- **PoT**: the model produces Python code. The answer is then calculated through the Python interpreter.
- **Combine**: the model produces Python code for questions that require calculation, and Direct answers for others. We determine whether a question requires calculation with a simple rule-based keyword detector. If the question contains one of the calculative keywords<sup>1</sup>, the detector will treat it as a computational question and prompt the model to generate a PoT answer. Otherwise, the model is instructed to produce a Direct answer. Additionally, if the generated program of a calculative question encounters syntax errors, we let the model produce Direct answers for this question in the Combine setting.
- **Oracle** We further introduce the Oracle setting for ChartQA evaluation. Under this setting, we always choose the correct one between the Direct and PoT answers after evaluating under both settings. It is the upper bound of the combination across the two answers.

We evaluate TinyChart under the Combine setting by default.

**Chart-to-Text** Chart-to-Text aims to generate a chart summarization based on chart content. We evaluate the model with the Pew benchmark [21], and report BLEU4 [41] as the metric.

**Chart-to-Table** Chart-to-Table aims to extract the underlying data table presented by the chart. We evaluate the performance of Chart-to-Table with the data table annotation provided by ChartQA [32] following [10, 35]. We report RMS<sub>F1</sub> [26] as the metric.

**OpenCQA** Different from ChartQA, OpenCQA [20] evaluates the ability of models to generate free-form answers to the chart-related questions. We report BLEU4 [41] as the metric following [34, 35].

**ChartX** ChartX [51] is a recently proposed benchmark that contains more chart types. We evaluate the ChartX cognition tasks since they are more challenging. It covers Question Answering, Chart Description Generation, Chart Summary Generation, and Chart Redrawing. We report the GPT-Accuracy for QA and GPT-score for the remaining 3 tasks as the metrics following ChartX [51].

### 4.3 Main Results

Table 2 shows an extensive comparison between TinyChart and existing multimodal large language models on 4 chart understanding benchmarks. Our TinyChart model achieves state-of-the-art performance on ChartQA, Chart-to-Text, Chart-to-Table, and OpenCQA, while excels in larger inference throughput. Specifically, with the input resolution set at 768×768, TinyChart achieves an accuracy of 83.60 on ChartQA [32], surpassing several closed-source models including GPT-4V, Gemini-Ultra, and Qwen-VL-Max [1]. It also outperforms previous open-source SOTA ChartAst [35] on chart understanding.

We find that previous models performed poorly on the ChartQA human subset, with none of them achieving over 70%. In contrast, the performance on the ChartQA-augmentation has approached 93.9%. This is because the questions posed by human annotators involve more computational problems [32] and are more challenging. By leveraging the Program-of-Thoughts learning, TinyChart achieves performance of 73.34% on ChartQA-human, which is an improvement of 7.44% over the previous state-of-the-art ChartAst [35]. This demonstrates the effectiveness of our proposed learning method based on the Program-of-Thoughts.

We observed that models with higher input resolutions generally perform better on chart understanding tasks. However, encoding

<sup>1</sup>sum, mean, average, ratio, mode, divide, dividing, differ, subtract, add, division, times, absolute, minus, exceed, below, less, fewer, bigger, biggest, greater, higher, longer, tallest, lowest, number, how many colors, what is the value

**Table 3: Performance on ChartQA under different settings.**

Model	ChartQA			
	Direct	PoT	Oracle	Combine
ChartLlama [10]	69.66	-	-	-
ChartAst [35]	75.10	-	-	79.90
TinyChart@512	<b>76.92</b>	79.64	88.76	82.88
TinyChart@768	76.36	<b>80.84</b>	<b>89.12</b>	<b>83.60</b>

**Table 4: ChartQA performance on Calculative (Cal.) and Non-calculative (Non-cal.) questions.**

Model	Setting	ChartQA		
		Cal. (761)	Non-cal. (1739)	Total (2500)
TinyChart@768	Direct	56.64	<b>84.99</b>	76.36
TinyChart@768	PoT	78.98	81.66	80.84
TinyChart@768	Combine	<b>80.42</b>	<b>84.99</b>	<b>83.60</b>

high-resolution charts leads to a decrease in inference speed (e.g., Qwen-VL vs. Llava1.5, DocOwl1.5 vs. UReader, ChartAst vs. ChartLlama). By leveraging visual token merging, TinyChart is able to accept higher-resolution input images with a limited increase in computing demands, thus achieving better performance. Due to the smaller model size and the efficient visual token merging strategy, TinyChart achieves significantly larger inference throughput compared to previous models. In summary, these results demonstrate that TinyChart can achieve efficient chart understanding with enhanced performance and faster inference.

**ChartQA performance under different settings.** Table 3 shows the performance comparison under different settings. Note that the performance of ChartAst under the Combine setting is from Meng et al. [35], which leverages a combination of Direct answer and executive JSON to produce the final answer. The results indicate that our TinyChart model could achieve SOTA performance on the Direct answer. By combining with PoT answers, TinyChart could make further improvements. In addition, since the combination of Direct and PoT answers is very simple, the performance under the Combine setting falls behind the Oracle setting a lot. Further study can be conducted to better combine the two answers.

**Calculative and non-calculative questions.** We divide the questions in ChartQA test set [32] into two categories: calculative questions (761 of 2500) and non-calculative questions (1739 of 2500) by checking whether they contain calculative keywords mentioned above. Table 4 shows the performance of TinyChart@768 on these two types of questions under different settings. We observe that PoT significantly improves the performance on calculative questions compared to Direct settings (78.98 vs. 56.64) and thus it shows overall performance gains (80.84 vs. 76.36). And the simple combination of Direct and PoT strategies further makes improvements.

**Evaluation on ChartX.** To further assess the generalizability of TinyChart, we compare our model with end-to-end General MLLM

**Table 5: Evaluation results on ChartX [51].**

Model	ChartX Cognition			
	QA	Summary	Description	Redrawing
<b>General MLLM</b>				
Llava1.5	17.19	1.48	1.29	0.75
GPT-4V	33.04	<b>3.17</b>	<b>3.12</b>	<b>2.63</b>
<b>Chart MLLM</b>				
ChartLlama	13.80	1.04	1.02	0.94
ChartAst	30.99	0.33	1.03	0.82
TinyChart@768	<b>33.35</b>	1.53	1.64	1.89

and Chart MLLM on ChartX-Cognition benchmark [51], since it covers visually diverse chart types. We use TinyChart@768 to perform inference on ChartX without additional fine-tuning. As shown in Table 5, benefiting from our Program-of-Thoughts learning method, TinyChart achieves a 33.35 GPT-Accuracy on the QA task, even surpassing the GPT-4V model. Though it falls behind GPT-4V in Summary, Description, and Redrawing tasks, TinyChart still performs better than open-source Chart MLLMs including ChartLlama and ChartAst. It indicates that TinyChart has a strong capability to generalize across various chart types.

#### 4.4 Ablation Studies

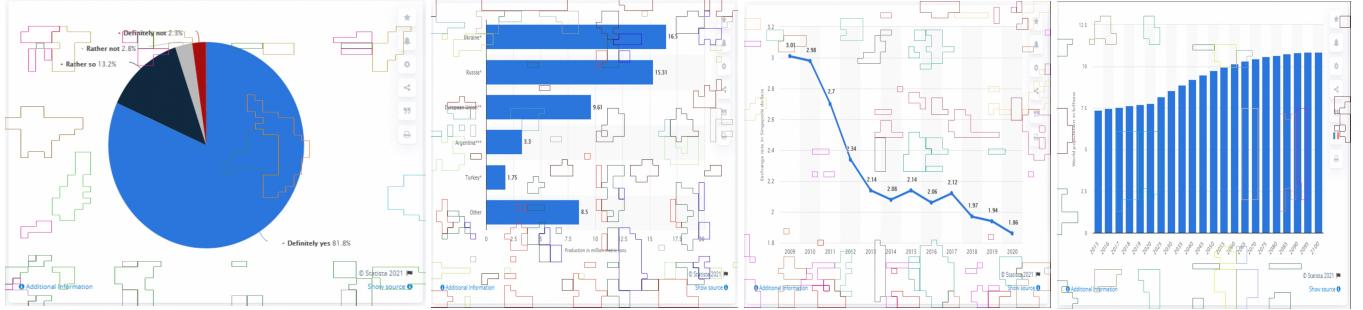
To verify the effectiveness of visual token merging and program-of-thoughts learning, we conduct ablation studies in Table 6.

**Ablation on PoT learning.** The upper block in Table 6 shows the performance of the model with and without the use of Program-of-Thoughts training data. Comparing Row 2 with Row 1, we observe that training solely with template-based PoT improves the model’s ability to generate direct answers (71.12 vs. 70.72). This improvement is attributed to PoT learning enhances the model’s reasoning abilities. At this point, the PoT answers produced by the model are less accurate than direct answers (55.44 vs. 71.12), which may be due to the inability of template-based PoT to cover all questions. However, when we ask the model to generate PoT answers for questions that require calculation and combine with direct answers, it outperforms solely direct answers (73.00 vs. 71.12). This indicates that PoT answers have advantages in computational problems. After incorporating GPT-based PoT into training, the performance of PoT answering surpasses direct answering (76.88 vs. 72.44), and both direct (72.44 vs. 71.12) and combined answering (79.48 vs. 73.00) show further improvements. These results confirm the effectiveness of our proposed Program-of-Thoughts learning method, suggesting that it not only strengthens the model’s computational capabilities but also enhances overall problem-solving capability.

**Ablation on Visual Token Merging.** The middle block in Table 6 compares the performance with and without using visual token merging when the input resolution is 512×512, and with different numbers of tokens to merge in each layer. Comparing Row 4 and Row 3, increasing the input resolution from 384 to 512 significantly improves the model’s performance on three chart understanding

**Table 6: Ablation study. We train the models only using benchmark datasets in this experiment.**

Row	Resolution	GPT	Template	Visual Patch	Visual	Inference	ChartQA		Chart2Text	Chart2Table	
		PoT	PoT	Merge	Length	Throughput	Direct	PoT	Combine	BLEU4	RMS <sub>F1</sub>
1	384×384	×	×	×	729	3.73 it/s	70.72	-	-	17.10	85.80
2	384×384	×	✓	×	729	3.73 it/s	71.12	55.44	73.00	17.04	87.68
3	384×384	✓	✓	×	729	3.73 it/s	72.44	76.88	79.48	16.67	87.30
4	512×512	✓	✓	×	1,296	2.38 it/s	74.08	79.64	81.72	17.32	89.76
5	512×512	✓	✓	r=12	984	2.84 it/s	73.24	77.72	80.52	16.54	88.26
6	512×512	✓	✓	r=15	906	3.26 it/s	72.52	78.60	80.04	16.96	88.01
7	512×512	✓	✓	r=20	776	3.65 it/s	73.36	78.84	80.76	16.57	87.81
8	768×768	✓	✓	×	2,916	OOM	-	-	-	-	-
9	768×768	✓	✓	r=84	732	3.14 it/s	73.24	77.72	81.04	16.43	88.90

**Figure 5: Visual token merging visualization. Top 10 groups with the most merged tokens are outlined in different colors.**

benchmarks, demonstrating that high resolution is crucial for comprehending chart images. However, a direct increase in resolution leads to a substantial drop in the inference throughput (2.38 it/s vs. 3.73 it/s). The reason is that, given high-resolution images, the standard vision transformer produces a lengthy visual feature sequence that is then processed by the large language model. This brings considerable computational expenses. By adopting the visual token merging, we can control the length of the visual feature sequence by regulating the number of tokens to merge at each layer, and, thereby achieving efficient high-resolution encoding. When setting  $r=20$ , we attain an inference throughput nearly equal to that with an input resolution of 384×384 (3.65 it/s vs. 3.73 it/s), while providing the performance benefits of higher resolutions.

**Extending to higher resolution.** To further highlight the advantages of visual token merging, we increase the input resolution to 768 in the bottom block of Table 6. At this point, the length of the visual feature sequence is 2,916, which could not be trained using 32GB V100 due to insufficient VRAM. However, after employing the visual token merging module with  $r=84$ , the input sequence length is reduced to 732 and we can perform training normally. In this setting, the model’s inference throughput is 3.14 it/s, and demonstrates a certain performance advantage in ChartQA (81.04 vs. 80.76) and Chart-to-Table (88.90 vs. 87.81). It illustrates that by utilizing visual token merging, we are able to leverage higher-resolution chart images under constrained resources, thereby improving performance.

## 4.5 Visualization

To investigate the effects of visual token merging, we visualized the token merging results at the final layer of the vision transformer. In Figure 5, we visualize the top ten groups with the largest numbers of tokens. Each group is outlined with a different color. The visualization reveals that these largest groups typically correspond to blank or colored areas. By compressing these areas down to a single token for encoding, our visual token merging module can thus reduce the length of the encoded sequence without losing much information, thereby achieving efficient visual encoding.

## 4.6 Case study

We conduct case studies with TinyChart when conducting chart question answering, chart-to-table, chart-to-text, and chart redrawing in Figure 6, 7, 8, and 9.

**Chart Question Answering.** In Figure 6, we present a case study on ChartQA. As shown in Figure 6 (a-c), much key information within the chart is provided by visually situated texts within the image, which requires the model to have the ability to process high-resolution images. Since ChartLlama only supports 336 resolutions, it struggles to retrieve accurate information in these charts. In contrast, thanks to the visual token merging, our TinyChart can accept higher-resolution inputs without introducing excessive computations. Thus it can successfully find clues related to the questions. Meanwhile, ChartLlama suffers from numerical errors

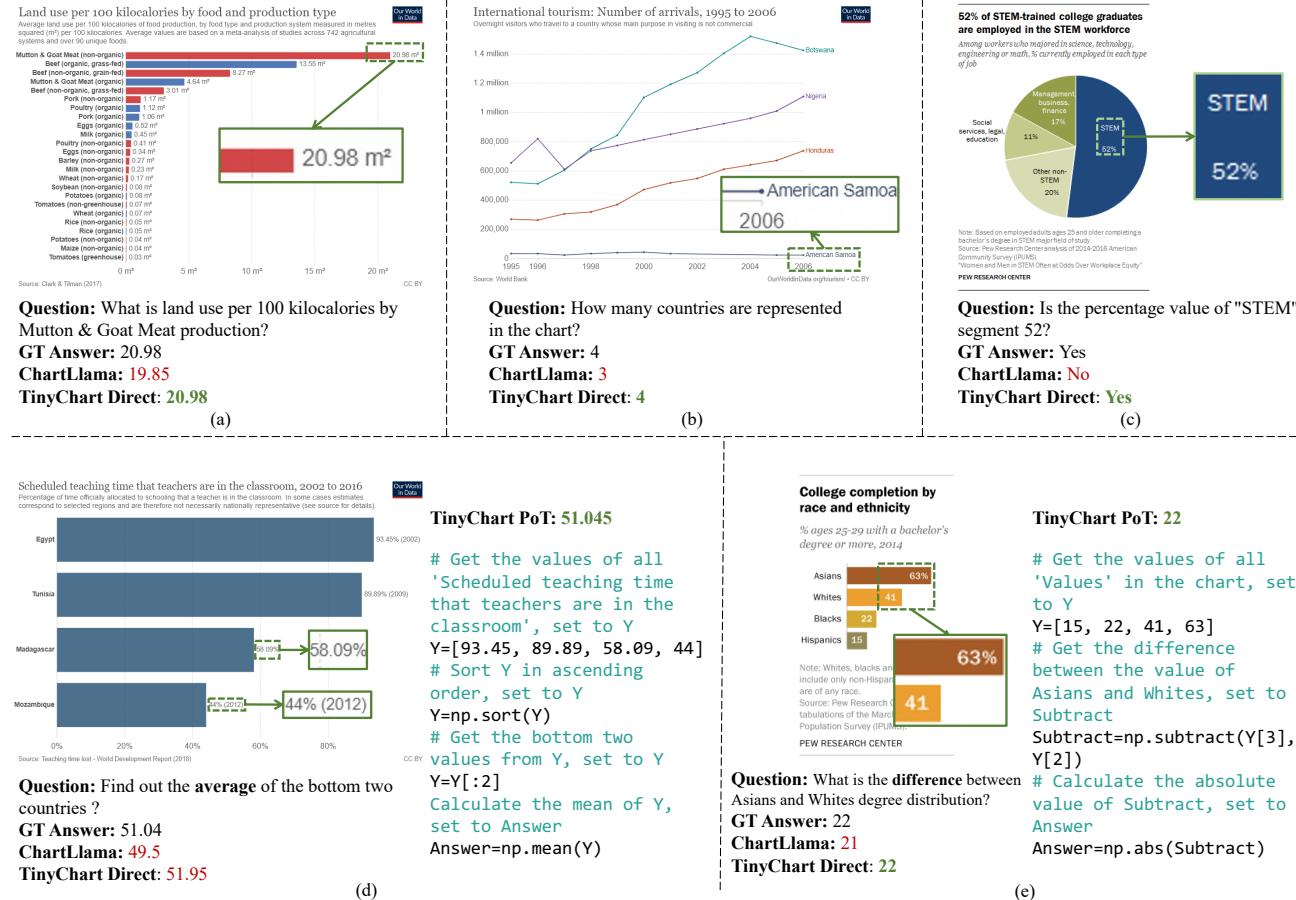


Figure 6: Case studies on ChartQA. We compare TinyChart@768 with ChartLlama.

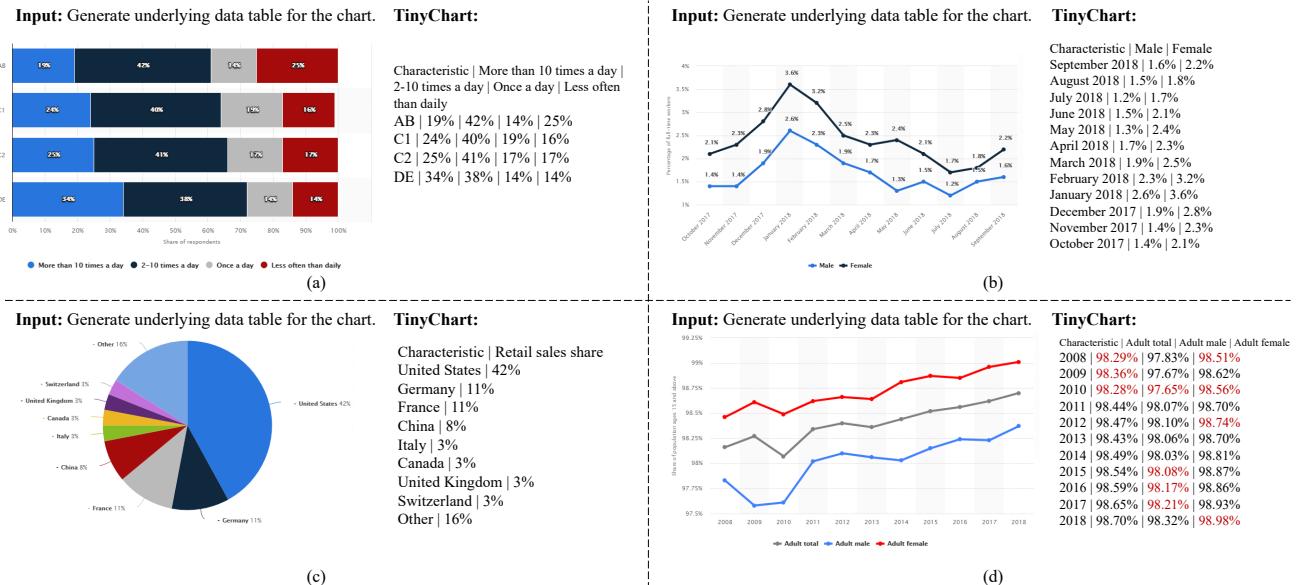
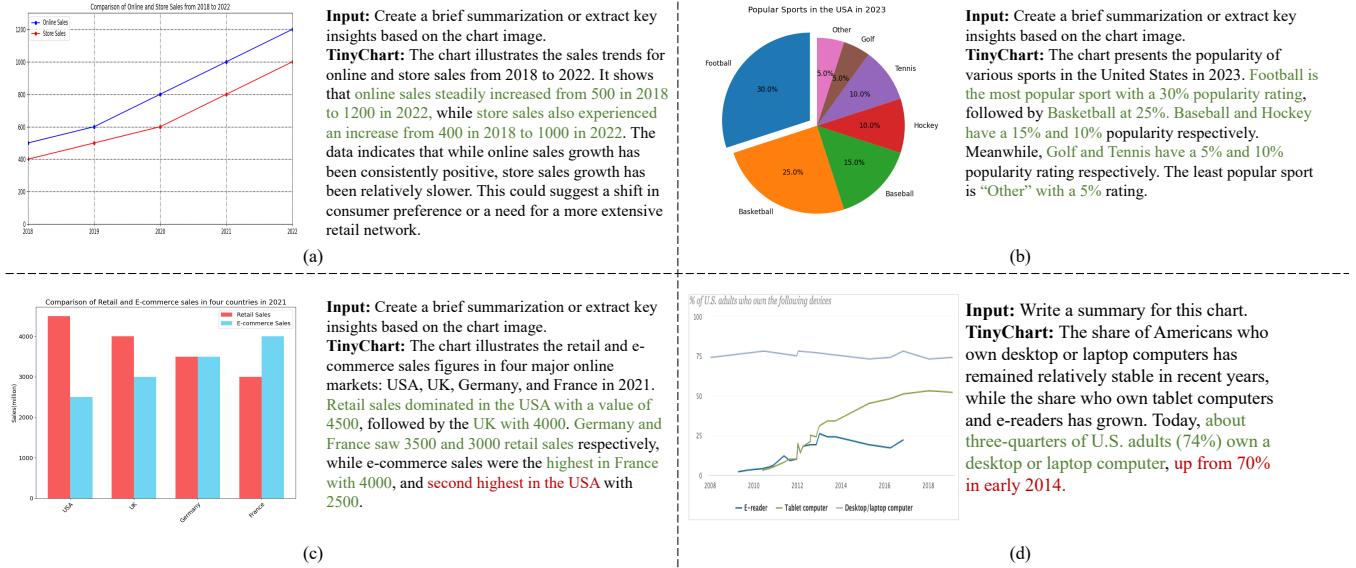
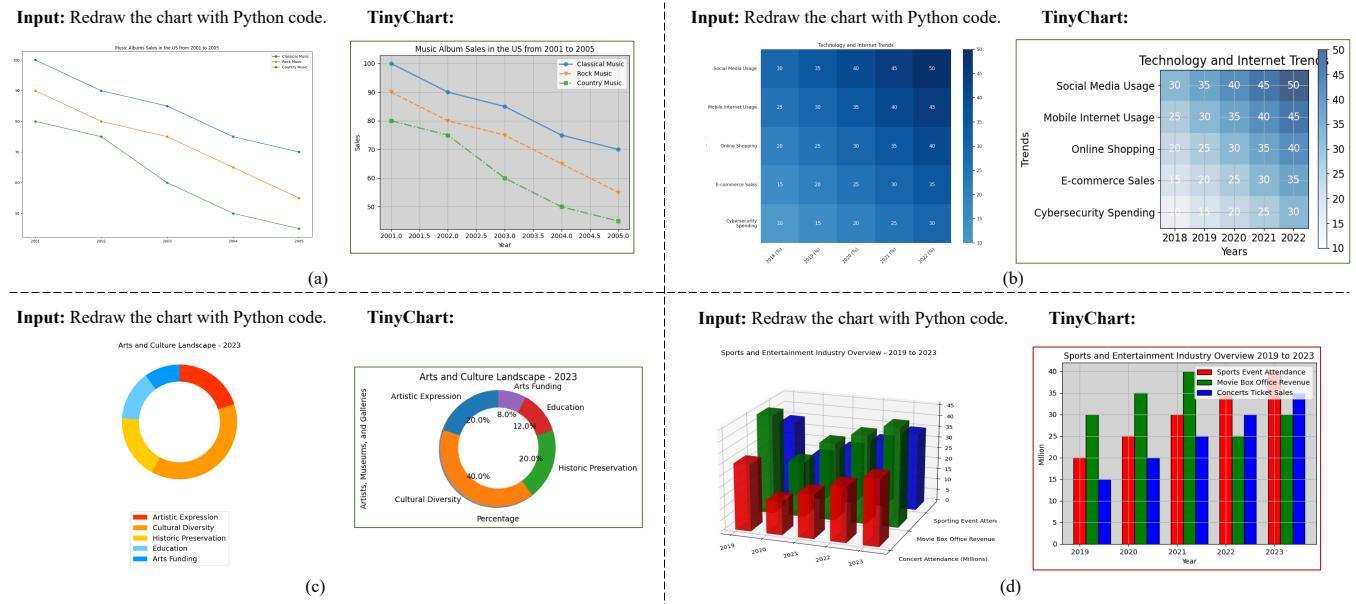


Figure 7: Examples of chart-to-table extraction of TinyChart@768. The wrong values produced by the model are marked red.



**Figure 8: Cases of chart-to-text generation by TinyChart@768. Correct contents are shown in green and wrong contents are marked red.**



**Figure 9: Examples of chart redrawing. We present the resulting image after executing the Python code produced by the model. The bad case is with the red bounding box.**

when faced with calculative questions in Figure 6 (d-e), and our PoT (Program-of-Thoughts) learning method can accurately solve these problems. These examples further illustrate the advantages of our methods.

**Chart-to-Table.** For chart-to-table extraction, we find that our TinyChart model can successfully extractive values from several visually diverse charts in Figure 7 (a-c), thanks to its excellent

text recognition ability with high-resolution input. However, as shown in Figure 7 (d), the model struggles to estimate the values of data points in the absence of OCR words. It seems that the model could make reasonable predictions based on surrounding points, but hardly provide accurate values based on the coordinate axis. This indicates that the model still lacks the ability to understand spatial relationships across large areas.

**Chart-to-Text.** From Figure 8, we observe that the model can understand the data presented in the chart and generate descriptions and summaries in natural language. Though it can retrieve the data values correctly, we find it sometimes produces contents that do not match the chart as shown in Figure 8 (c-d). This may be due to the inherent limitations of hallucination in MLLMs [24, 43, 49, 50], and may be alleviated by addressing hallucinations [17, 18, 22, 56].

**Chart redrawing.** We present four cases of chart redrawing in Figure 9. As shown in Figure 9 (a-c), our TinyChart model can generate Python code to redraw visually diverse chart types including lines, heatmaps, and rings. However, it can be hard to draw unseen chart types such as 3D bar charts (Figure 9 (d)). This may be mitigated by improving the coverage of different chart types in training data through automatic data construction techniques [10, 51].

## 5 CONCLUSION

This paper introduces TinyChart, a chart understanding Multimodal Large Language Model with 3 billion parameters. To address the inefficiency of lengthy visual token sequences with high-resolution images, TinyChart injects a visual token merging module that merges similar vision tokens together, thereby enabling efficient encoding of high-resolution images. To tackle the challenges of learning numerical computations, we propose a Program-of-Thoughts learning method that trains the model to generate Python programs to answer questions. Our TinyChart model achieves state-of-the-art (SOTA) performance on multiple chart understanding benchmarks, surpassing existing 13 billion parameter chart MLLMs, and outperforms closed-source models like GPT-4V on ChartQA. Extensive ablation studies confirm the effectiveness of our methods. Our code and model are released at <https://github.com/X-PLUG/mPLUG-DocOwl/tree/main/TinyChart>.

## REFERENCES

- [1] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-VL: A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond. arXiv:2308.12966 [cs.CV]
- [2] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. 2023. Token Merging: Your ViT But Faster. In *The Eleventh International Conference on Learning Representations*. <https://openreview.net/forum?id=JroZRaRw7Eu>
- [3] Jinyue Chen, Lingyu Kong, Haoran Wei, Chenglong Liu, Zheng Ge, Liang Zhao, Jianjian Sun, Chunrui Han, and Xiangyu Zhang. 2024. OneChart: Purify the Chart Structural Extraction via One Auxiliary Token. arXiv:2404.09987 [cs.CV]
- [4] Wenhua Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research* (2023).
- [5] Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Bin Wang, Linke Ouyang, Xilin Wei, Songyang Zhang, Haodong Duan, Maosong Cao, et al. 2024. InternLM-XComposer2: Mastering free-form text-image composition and comprehension in vision-language large model. arXiv preprint arXiv:2401.16420 (2024).
- [6] Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Bin Wang, Linke Ouyang, Songyang Zhang, Haodong Duan, Wenwei Zhang, Yining Li, et al. 2024. InternLM-XComposer2-4KHD: A Pioneering Large Vision-Language Model Handling Resolutions from 336 Pixels to 4K HD. arXiv preprint arXiv:2404.06512 (2024).
- [7] 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. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR* abs/2010.11929 (2020). arXiv:2010.11929 <https://arxiv.org/abs/2010.11929>
- [8] Hao Feng, Qi Liu, Hao Liu, Wengang Zhou, Houqiang Li, and Can Huang. 2023. DocPedia: Unleashing the Power of Large Multimodal Model in the Frequency Domain for Versatile Document Understanding. arXiv preprint arXiv:2311.11810 (2023).
- [9] Jiayun Fu, Bin B Zhu, Haidong Zhang, Yayı Zou, Song Ge, Weiwei Cui, Yun Wang, Dongmei Zhang, Xiaojing Ma, and Hai Jin. 2022. Chartstamp: Robust chart embedding for real-world applications. In *Proceedings of the 30th ACM International Conference on Multimedia*. 2786–2795.
- [10] Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. 2023. ChartLlama: A multimodal llm for chart understanding and generation. arXiv preprint arXiv:2311.16483 (2023).
- [11] Dan Hendrycks and Kevin Gimpel. 2016. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR* abs/1606.08415 (2016). arXiv:1606.08415 <http://arxiv.org/abs/1606.08415>
- [12] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 2023. Cogagent: A visual language model for gui agents. arXiv preprint arXiv:2312.08914 (2023).
- [13] Anwen Hu, Shizhe Chen, and Qin Jin. 2021. Question-controlled Text-aware Image Captioning. In *Proceedings of the 29th ACM International Conference on Multimedia* (Virtual Event, China) (MM ’21). Association for Computing Machinery, New York, NY, USA, 3097–3105. <https://doi.org/10.1145/3474085.3475452>
- [14] Anwen Hu, Yaya Shi, Haiyang Xu, Jiabo Ye, Qinghao Ye, Ming Yan, Chenliang Li, Qi Qian, Ji Zhang, and Fei Huang. 2024. mPLUG-PaperOwl: Scientific Diagram Analysis with the Multimodal Large Language Model. arXiv:2311.18248 [cs.MM]
- [15] Anwen Hu, Haiyang Xu, Jiabo Ye, Ming Yan, Liang Zhang, Bo Zhang, Chen Li, Ji Zhang, Qin Jin, Fei Huang, and Jingren Zhou. 2024. mPLUG-DocOwl 1.5: Unified Structure Learning for OCR-free Document Understanding. arXiv:2403.12895 [cs.CV]
- [16] Kung-Hsiang Huang, Hou Pong Chan, Yi R. Fung, Haoyi Qiu, Mingyang Zhou, Shafiq Joty, Shih-Fu Chang, and Heng Ji. 2024. From Pixels to Insights: A Survey on Automatic Chart Understanding in the Era of Large Foundation Models. arXiv:2403.12027 [cs.CL]
- [17] Qidong Huang, Xiaoyi Dong, Pan Zhang, Bin Wang, Conghui He, Jiaqi Wang, Dahua Lin, Weiming Zhang, and Nenghai Yu. 2024. OPERA: Alleviating Hallucination in Multi-Modal Large Language Models via Over-Trust Penalty and Retrospection-Allocation. arXiv:2311.17911 [cs.CV]
- [18] Chaoya Jiang, Haiyang Xu, Mengfan Dong, Jiaxing Chen, Wei Ye, Ming Yan, Qinghao Ye, Ji Zhang, Fei Huang, and Shikun Zhang. 2024. Hallucination-Augmented Contrastive Learning for Multimodal Large Language Model. arXiv:2312.06968 [cs.CV]
- [19] Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. 2018. Dvqa: Understanding data visualizations via question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5648–5656.
- [20] Shankar Kantharaj, Xuan Long Do, Rixie Tiffany Leong, Jia Qing Tan, Enamul Hoque, and Shafiq Joty. 2022. OpenCQA: Open-ended Question Answering with Charts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 11817–11837. <https://doi.org/10.18653/v1/2022.emnlp-main.811>
- [21] Shankar Kantharaj, Rixie Tiffany Leong, Xiang Lin, Ahmed Masry, Megh Thakkar, Enamul Hoque, and Shafiq Joty. 2022. Chart-to-Text: A Large-Scale Benchmark for Chart Summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 4005–4023. <https://doi.org/10.18653/v1/2022.acl-long.277>
- [22] Sicong Leng, Hang Zhang, Guanzheng Chen, Xin Li, Shijian Lu, Chunyan Miao, and Lidong Bing. 2023. Mitigating Object Hallucinations in Large Vision-Language Models Through Visual Contrastive Decoding. arXiv:2311.16922 [cs.CV]
- [23] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023. Textbooks Are All You Need II: phi-1.5 technical report. arXiv:2309.05463 [cs.CL]
- [24] Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Evaluating Object Hallucination in Large Vision-Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 292–305.
- [25] Ziyi Lin, Chris Liu, Renrui Zhang, Peng Gao, Longtian Qiu, Han Xiao, Han Qiu, Chen Lin, Wenqi Shao, Kegin Chen, Jiaming Han, Siyuan Huang, Yichi Zhang, Xuming He, Hongsheng Li, and Yu Qiao. 2023. SPHINX: The Joint Mixing of Weighted Tasks, and Visual Embeddings for Multi-modal Large Language Models. arXiv:2311.07575 [cs.CV]
- [26] Fangyu Liu, Julian Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhua Chen, Nigel Collier, and Yasemin Altun. 2023. DePlot: One-shot visual language reasoning by plot-to-table translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 10381–10399. <https://doi.org/10.18653/v1/2023.findings-acl.660>
- [27] Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Eisenschlos. 2023. MatCha: Enhancing Visual Language Pretraining with Math Reasoning and Chart Derendering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber,

- and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 12756–12770. <https://doi.org/10.18653/v1/2023.acl-long.714>
- [28] Fuxiao Liu, Xiaoyang Wang, Wenlin Yao, Jianshu Chen, Kaiqiang Song, Sangwoo Cho, Yaser Yacoob, and Dong Yu. 2023. Mmc: Advancing multimodal chart understanding with large-scale instruction tuning. *arXiv preprint arXiv:2311.10724* (2023).
- [29] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023. Improved Baselines with Visual Instruction Tuning. *arXiv:2310.03744* [cs.CV]
- [30] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024. Visual instruction tuning. *Advances in neural information processing systems* 36 (2024).
- [31] Yuliang Liu, Zhang Li, Biao Yang, Chunyuan Li, Xucheng Yin, Cheng lin Liu, Lianwan Jin, and Xiang Bai. 2024. On the Hidden Mystery of OCR in Large Multimodal Models. *arXiv:2305.07895* [cs.CV]
- [32] Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. ChartQA: A Benchmark for Question Answering about Charts with Visual and Logical Reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*. 2263–2279.
- [33] Ahmed Masry, Parsa Kavehzadeh, Xuan Long Do, Enamul Hoque, and Shafiq Joty. 2023. UniChart: A Universal Vision-language Pretrained Model for Chart Comprehension and Reasoning. *arXiv:2305.14761* [cs.CL]
- [34] Ahmed Masry, Mehrad Shahmohammadi, Md Rizwan Parvez, Enamul Hoque, and Shafiq Joty. 2024. ChartInstruct: Instruction Tuning for Chart Comprehension and Reasoning. *arXiv:2403.09028* [cs.CL]
- [35] Fanqing Meng, Wenqi Shao, Quanfeng Lu, Peng Gao, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024. ChartAssistant: A Universal Chart Multimodal Language Model via Chart-to-Table Pre-training and Multitask Instruction Tuning. *arXiv preprint arXiv:2401.02384* (2024). *arXiv:2401.02384*
- [36] Nitesh Methani, Pritha Ganguly, Mitesh M Khapra, and Pratyush Kumar. 2020. Plotqa: Reasoning over scientific plots. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 1527–1536.
- [37] Jason Obeid and Enamul Hoque. 2020. Chart-to-Text: Generating Natural Language Descriptions for Charts by Adapting the Transformer Model. *CoRR abs/2010.09142* (2020). *arXiv:2010.09142* <https://arxiv.org/abs/2010.09142>
- [38] OpenAI. 2023. GPT-3.5-Turbo. <https://platform.openai.com/docs/models/gpt-3-5-turbo>.
- [39] OpenAI. 2023. GPT-4 Technical Report. *arXiv:2303.08774* [cs.CL]
- [40] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.
- [41] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.
- [42] Raian Rahman, Rizvi Hasan, Abdullah Al Farhad, Md. Tahmid Rahman Laskar, Md. Hamajul Ashmafee, and Abu Raihan Mostofa Kamal. 2023. ChartSumm: A Comprehensive Benchmark for Automatic Chart Summarization of Long and Short Summaries. *Proceedings of the Canadian Conference on Artificial Intelligence* (jun 5 2023). <https://cailac.pubpub.org/pub/ujhjycsw>.
- [43] Anna Rohrbach, Lisa Anne Hendricks, Kaylee Burns, Trevor Darrell, and Kate Saenko. 2018. Object Hallucination in Image Captioning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 4035–4045.
- [44] Amarnpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. 2019. Towards VQA Models That Can Read. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [45] Benny Tang, Angie Boggust, and Arvind Satyanarayanan. 2023. VisText: A Benchmark for Semantically Rich Chart Captioning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 7268–7298. <https://doi.org/10.18653/v1/2023.acl-long.401>
- [46] Gemini Team, Rohan Anil, Sébastien Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).
- [47] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. 2011. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering* 13, 2 (2011), 22–30. <https://doi.org/10.1109/MCSE.2011.37>
- [48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
- [49] Junyang Wang, Yuhang Wang, Guohai Xu, Jing Zhang, Yukai Gu, Haitao Jia, Ming Yan, Ji Zhang, and Jitao Sang. 2023. An llm-free multi-dimensional benchmark for mlmns hallucination evaluation. *arXiv preprint arXiv:2311.07397* (2023).
- [50] Junyang Wang, Yiyang Zhou, Guohai Xu, Pengcheng Shi, Chenlin Zhao, Haiyang Xu, Qinghao Ye, Ming Yan, Ji Zhang, Jihua Zhu, Jitao Sang, and Haoyu Tang. 2023. Evaluation and Analysis of Hallucination in Large Vision-Language Models. *arXiv:2308.15126* [cs.LG]
- [51] Renqiu Xia, Bo Zhang, Hancheng Ye, Xiangchao Yan, Qi Liu, Hongbin Zhou, Zijun Chen, Min Dou, Botian Shi, Junchi Yan, and Yu Qiao. 2024. ChartX & ChartVLM: A Versatile Benchmark and Foundation Model for Complicated Chart Reasoning. *arXiv:2402.12185* [cs.CV]
- [52] Jiabo Ye, Anwen Hu, Haiyang Xu, Qinghao Ye, Ming Yan, Guohai Xu, Chenliang Li, Junfeng Tian, Qi Qian, Ji Zhang, Qin Jin, Liang He, Xin Lin, and Fei Huang. 2023. UReader: Universal OCR-free Visually-situated Language Understanding with Multimodal Large Language Model. In *EMNLP (Findings)*. Association for Computational Linguistics, 2841–2858.
- [53] Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, Chenliang Li, Yunhong Xu, Hehong Chen, Junfeng Tian, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. 2024. mPLUG-Owl: Modularization Empowers Large Language Models with Multimodality. *arXiv:2304.14178* [cs.CL]
- [54] Qinghao Ye, Haiyang Xu, Ming Yan, Chenlin Zhao, Junyang Wang, Xiaoshan Yang, Ji Zhang, Fei Huang, Jitao Sang, and Changsheng Xu. 2023. mPLUG-Octopus: The Versatile Assistant Empowered by A Modularized End-to-End Multimodal LLM. In *Proceedings of the 31st ACM International Conference on Multimedia*. 9365–9367.
- [55] Qinghao Ye, Haiyang Xu, Ming Yan, Anwen Hu, Haowei Liu, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. 2023. mPLUG-Owl2: Revolutionizing Multi-modal Large Language Model with Modality Collaboration. *arXiv:2311.04257* [cs.CL]
- [56] Zihao Yue, Liang Zhang, and Qin Jin. 2024. Less is More: Mitigating Multimodal Hallucination from an EOS Decision Perspective. *arXiv preprint arXiv:2402.14545* (2024).
- [57] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. 2023. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 11975–11986.
- [58] Liang Zhang, Anwen Hu, Jing Zhang, Shuo Hu, and Qin Jin. 2023. MPMQA: multimodal question answering on product manuals. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 13958–13966.
- [59] Pan Zhang, Xiaoyi Dong Bin Wang, Yuhang Cao, Chao Xu, Linke Ouyang, Zhiyuan Zhao, Shuangrui Ding, Songyang Zhang, Haodong Duan, Hang Yan, et al. 2023. Internlm-xcomposer: A vision-language large model for advanced text-image comprehension and composition. *arXiv preprint arXiv:2309.15112* (2023).
- [60] Peng Zhang, Yunlu Xu, Zhanzhan Cheng, Shiliang Pu, Jing Lu, Liang Qiao, Yi Niu, and Fei Wu. 2020. TRIE: end-to-end text reading and information extraction for document understanding. In *Proceedings of the 28th ACM International Conference on Multimedia*. 1413–1422.
- [61] Baichuan Zhou, Ying Hu, Xi Weng, Junlong Jia, Jie Luo, Xien Liu, Ji Wu, and Lei Huang. 2024. TinyLLaVA: A Framework of Small-scale Large Multimodal Models. *arXiv:2402.14289* [cs.LG]

## A CHARTQA-POT DETAILS

### A.1 Dataset Statistic

We build ChartQA-PoT based on the images and questions in the training split of ChartQA [32]. ChartQA-PoT consists of two subsets: Template-based PoT and GPT-based PoT. We present the statistics over ChartQA-PoT in Table 7. We find that answers provided by gpt-3.5-turbo are longer than template-based PoT, since they cover more diverse scenarios.

**Table 7: Statistic over ChartQA-PoT**

Statistic	Template PoT	GPT PoT	ChartQA PoT
Num. of samples	119,281	21,303	140,584
Num. of images	17,498	15,521	18,133
Avg. answer characters	319.38	381.23	328.75
Avg. answer tokens	117.70	136.01	120.48

We further present the first 2-gram words of the questions after removing stop words in Template-based PoT and GPT-based PoT in Figure 11. It is observed that GPT-PoT covers more diverse questions

### Instructions to gpt-3.5-turbo

Please generate a list of **assignment statements** in Python to answer the question of a chart. You can only use the following operators in each statement: <function\_list><sup>a</sup>. Do not use any circulation or if-branch. Do not include any unnecessary statement that is not used. The chart is presented by a data table with color information. Note that the colors are estimated and may not match the description in the question. You can choose the most possible data if necessary. You must provide a one-line comment before each assignment statement. The last variable must be Answer. Here are some examples:

#### Example Input #1:

Chart title: Long-term price index in food commodities, 1850-2015, World, 1934

Chart type: Horizontal bar chart

Chart table:

Food   Long-term price index in food commodities, 1850-2015, World, 1934
----- -----
Lamb (color: steelblue)   103.7
Corn (color: sienna)   103.13
Barley (color: mediumvioletred)   102.46
Rye (color: tomato)   87.37
Beef (color: sienna)   85.27
Wheat (color: slategray)   83.73

Question: What is the sum of the price index that is greater than 100?

Answer: 309.29

#### Example Output #1:

```
# Get the values of all 'Long-term price index of each food', set to Y
```

```
Y=[103.7, 103.13, 102.46, 87.37, 85.27, 83.73]
```

```
# Check whether Y is greater than 100, set to Greater
```

```
Greater=np.greater(Y,100)
```

```
# Find the indices where Greater is True, set to Indices
```

```
Indices=np.where(Greater)[0]
```

```
# Get the values at position Indices, set to Y
```

```
Y=np.array(Y)[Indices]
```

```
# Calculate the sum of all elements in Y, set to Answer
```

```
Answer=np.sum(Y)
```

**Input:** <target\_input>

**Output:**

```
"function_list=['len', 'all', 'any', 'index', 'np.sort', 'np.abs', 'np.add', 'np.argmax', 'np.argmin', 'np.diff', 'np.divide', 'np.greater', 'np.greater_equal', 'np.less', 'np.max', 'np.mean', 'np.median', 'np.min', 'np.subtract', 'np.sum', 'np.count_nonzero', 'np.where', '+', '^', '**', '/', '>', '<', '=']"
```

Figure 10: Instructions used for generating GPT-based PoT.

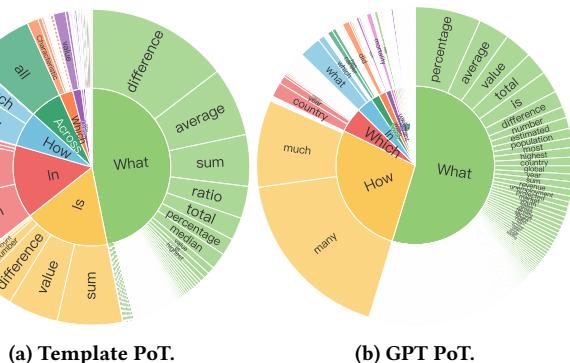


Figure 11: First 2-gram of the questions in ChartQA-PoT after removing stop words.

for ‘what’ type questions, and questions in Template-based PoT are more evenly distributed across all question types.

## A.2 Instructions for GPT-based PoT

Figure 10 shows the instructions for constructing GPT-based PoT answers. Note that we prompt gpt-3.5-turbo to provide Python code consisting of assignment statements and avoid using loops or judgment statements. This can simplify the program and reduce syntax errors. We also provide meta information including the chart title, type, and colors to gpt-3.5-turbo since some questions rely on this information to answer.