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

Liang Zhang[1*], **Anwen Hu**[2*], **Haiyang Xu**[2], **Ming Yan**[2],
Yichen Xu[1], **Qin Jin**[1†], **Ji Zhang**[2], **Fei Huang**[2],
[1]School of Information, Renmin University of China, [2]Alibaba Group
{zhangliang00,qjin,xu_yichen}@ruc.edu.cn
{huanwen.haw,shuofeng.xhy,ym119608,zj122146,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 chart understanding. However, the sheer size of these models 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 Program-of-Thoughts (PoT) learning, which trains the model to generate Python programs for numerical calculations, and (2) reduce lengthy vision feature sequences through Vision Token Merging, which gradually merges most similar vision tokens. Extensive experiments demonstrate that our 3B TinyChart achieves SOTA performance on various chart understanding benchmarks including ChartQA, Chart-to-Text, Chart-to-Table, OpenCQA, and ChartX. It outperforms several chart-understanding MLLMs with up to 13B parameters, and close-sourced MLLM GPT-4V on ChartQA, with higher throughput during inference due to a smaller model scale and more efficient vision encoding.

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 (Huang et al., 2024a). Automatic chart understanding received increasing attention from the research community (Han et al., 2023; Meng et al., 2024; Masry et al., 2024; Chen et al., 2024a). Recently, Multimodal Large Language Models (MLLMs) have shown strong capability in comprehending images and following instructions (OpenAI, 2023b; Liu et al., 2024a; Ye et al., 2024; Liu et al., 2023d; Lin et al., 2023; Ye et al., 2023c; Zhang et al., 2023b; Dong et al., 2024a,b). Based on MLLMs, some recent works (Han et al., 2023; Meng et al., 2024; Masry et al., 2024; Hu et al., 2024a) further build chart understanding models by constructing versatile chart comprehension datasets and performing supervised fine-tuning, They achieve significant performance increase in chart understanding benchmarks (Masry et al., 2022; Kantharaj et al., 2022b).

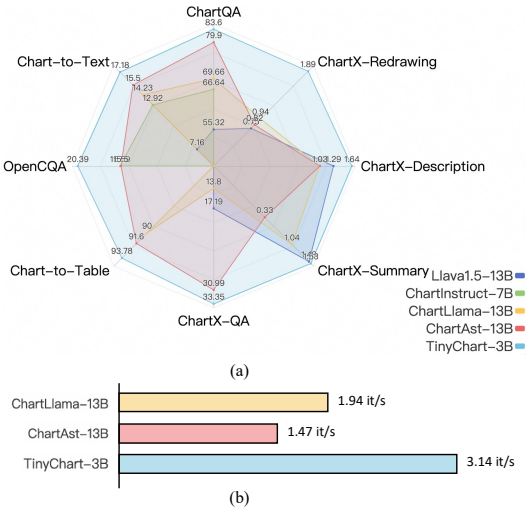


Figure 1: Our TinyChart-3B outperforms several 13B MLLMs on a variety of chart understanding benchmarks (a), while achieving larger inference throughput (b).

Despite their success, current chart understanding models still face three main limitations: (1) Considerable amount of parameters makes training and deployment challenging. For example, ChartLlama (Han et al., 2023) has 13 billion parameters, which is hard to deploy on a single GPU with less than 26 GB of VRAMs. (2) They are prone to errors when tackling questions involving numerical calculations (Meng et al., 2024), which are difficult to answer directly without any reasoning

[*] Equal contribution.
[†] Corresponding author.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 1882–1898
November 12-16, 2024

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 Bolya et al. (2023), 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, This enables controllable computation load when encoding high-resolution images.

To learn chart understanding more efficiently, inspired by Chen et al. (2023), we propose Program-of-Thoughts (PoT) learning that trains the model to generate Python programs for the computation problems step by step. The programs are then passed to a Python interpreter to produce final answers. To support PoT learning, we further construct the ChartQA-PoT dataset based on ChartQA (Masry et al., 2022). The QA pairs in ChartQA-PoT are constructed in two ways: (1) Template-based PoT, which generates questions and programs by filling in manually written templates with chart data. (2) GPT-based PoT, which leverages `gpt-3.5-turbo` (OpenAI, 2023a) to produce programs for human-written questions. Experimental results show that PoT 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. It outperforms several 13B MLLMs and achieves state-of-the-art performances on various chart understanding benchmarks, while excelling in inference speed.
- 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 to support PoT learning.
- 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

Chart Understanding requires the model to comprehend chart contents and accomplish related tasks such as data extraction (Liu et al., 2023a), QA (Masry et al., 2022; Methani et al., 2020; Kafle et al., 2018), summarization (Kantharaj et al., 2022b; Obeid and Hoque, 2020), and re-rendering (Han et al., 2023). It demands robust text recognition capabilities and computational reasoning from the model (Meng et al., 2024). Early approaches (Singh et al., 2019; Methani et al., 2020; Liu et al., 2023a; Fu et al., 2022; Zhang et al., 2023a; Hu et al., 2021) rely on off-the-shelf OCR tools to transform charts into textual representations such as data tables, and use language models to complete specified tasks. These approaches suffer from error accumulation since can not be optimized jointly.

Recent studies (Masry et al., 2023; Liu et al., 2023b; Han et al., 2023; Meng et al., 2024; Masry et al., 2024; Liu et al., 2023c) have shifted towards end-to-end methods based on MLLMs. They enhance MLLMs' (Liu et al., 2024a, 2023d; Ye et al., 2024, 2023c; Lin et al., 2023) chart understanding abilities through supervised fine-tuning (Ouyang et al., 2022) on chart instruction data (Han et al., 2023; Meng et al., 2024; Masry et al., 2024). Although these models achieves superior performance, the extensive model 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 SOTA performance on chart understanding.

Meanwhile, it is observed that MLLMs are prone to numerical errors (Masry et al., 2024; Meng et al., 2024). To resolve this, Meng et al. (2024) construct template-based executable command lines for numerical calculations. Their usage is constrained by the specific template and backend. In contrast, we train the model to generate Python code, which is more versatile. Also, we include GPT-generated programs from human-written questions in training data, which further improves the coverage of our method to different questions.

Multimodal Large Language Models (MLLMs) exhibit strong capabilities in visual understanding and instruction following (OpenAI, 2023b; Team et al., 2023). These models are generally trained on extensive general image-text data for cross-modal alignment and instruction fine-tuning (Liu et al., 2024a, 2023d; Li et al., 2024; Zhou et al., 2024; Zhang et al., 2023b; Dong et al., 2024a; Ye et al., 2023b; Chen et al., 2024b). Some studies (Liu et al., 2024b; Zhang et al., 2020) demonstrate a degree of OCR capability in these MLLMs, their performance on document and chart understanding benchmarks remains sub-optimal due to their low input resolution (Ye et al., 2023a; Dong et al., 2024b). Efforts in the general document domain have attempted to improve the fine-grained understanding capabilities of MLLMs by increasing resolution (Bai et al., 2023), segmenting images (Ye et al., 2023a; Hu et al., 2024b; Dong et al., 2024b), utilizing frequency domain signals (Feng et al., 2023), and introducing additional high-resolution encoders (Hong et al., 2023). 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 effectively reduce the length of visual sequences and reduce the computational cost of processing high-resolution input.

3 TinyChart

3.1 Model Architecture

Figure 2 shows the overview of TinyChart. It 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.

Vision Transformer Encoder process chart images into vision features. A standard vision transformer (Dosovitskiy et al., 2020) 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 (Vaswani et al., 2017). 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 feature would be $\lfloor \frac{N}{P} \rfloor^2$. In practice, when N is large, the vision feature sequence 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 (Hu et al., 2024b), 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 chart understanding, we apply Visual Token Merging (Bolya et al., 2023) in each transformer layer. The process of Visual Token Merging is shown in Figure 3. By merging the r most similar token 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 (Bolya et al., 2023). As shown in the lower part of Figure 3, the merging process 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 non-adjacent tokens can also be merged if they belong to different subsets.

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 address this, 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 (Vaswani et al., 2017). By adding $\log s$ inside softmax, the token that merged from s patches are duplicated by s times in the attention calculation (Bolya et al., 2023).

Vision Language Connector projects the vision features into the embedding space of the large language model. Following (Liu et al., 2023d; Zhou et al., 2024), we implement the vision-language connector as a MLP with one hidden layer and GeLU (Hendrycks and Gimpel, 2016) activation.

Large Language Model comprehend both visual features and language instructions, and then generate responses to accomplish chart understanding tasks. It is implemented as a transformer de-

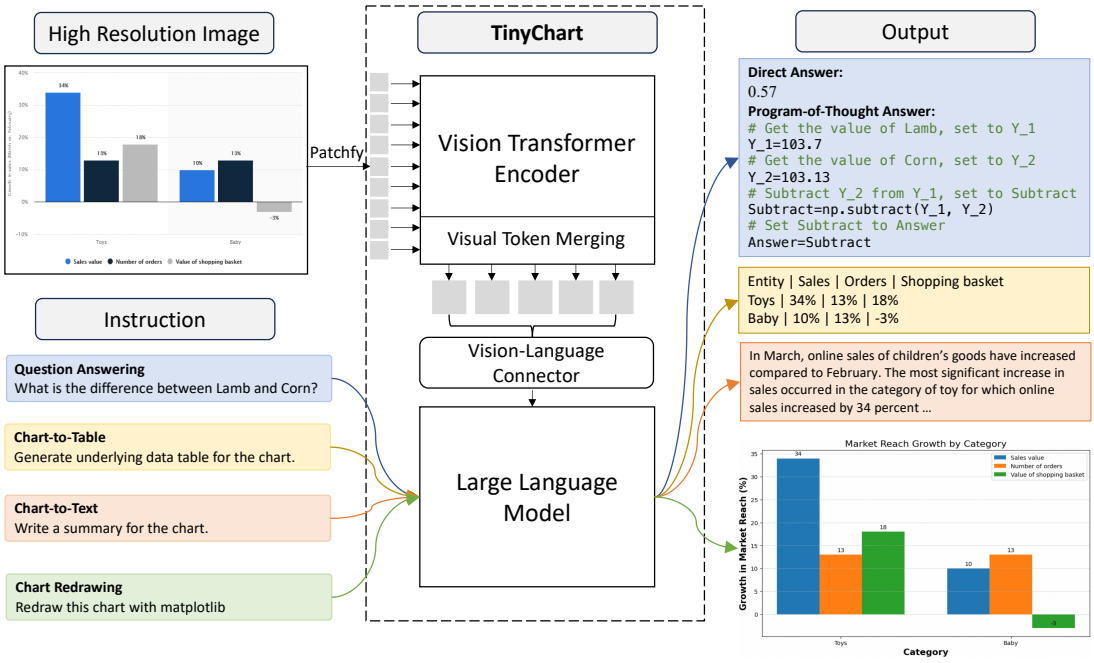


Figure 2: Overview of TinyChart.

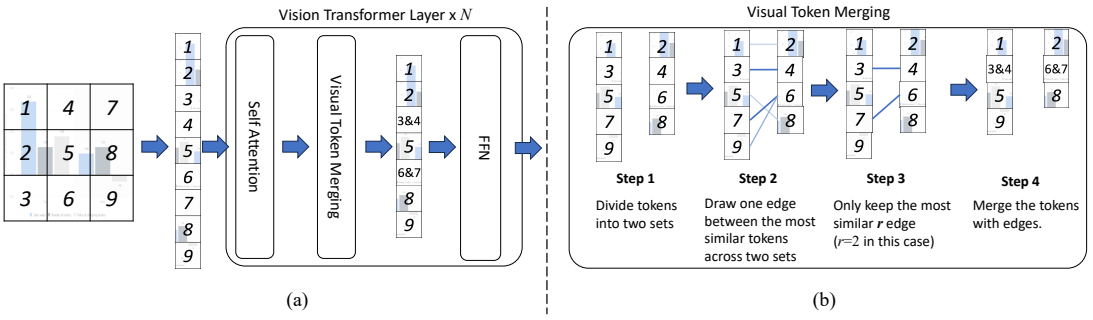


Figure 3: (a) Vision transformer layer with Visual Token Merging. (b) Process of the Visual Token Merging.

coder (Vaswani et al., 2017) 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:

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

Where T is the number of tokens in R. Note that we only calculate loss over response parts following (Liu et al., 2023d).

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 Python codes, whose execute results are answers of given questions. Compared to short answers that only contain the calculated values, the Python code includes comments and multi-step reasoning processes, which is easy to produce by large language models.

ChartQA-PoT Dataset To support PoT learning on chart understanding, we construct the ChartQA-PoT dataset based on the training split of ChartQA (Masry et al., 2022). 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.

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

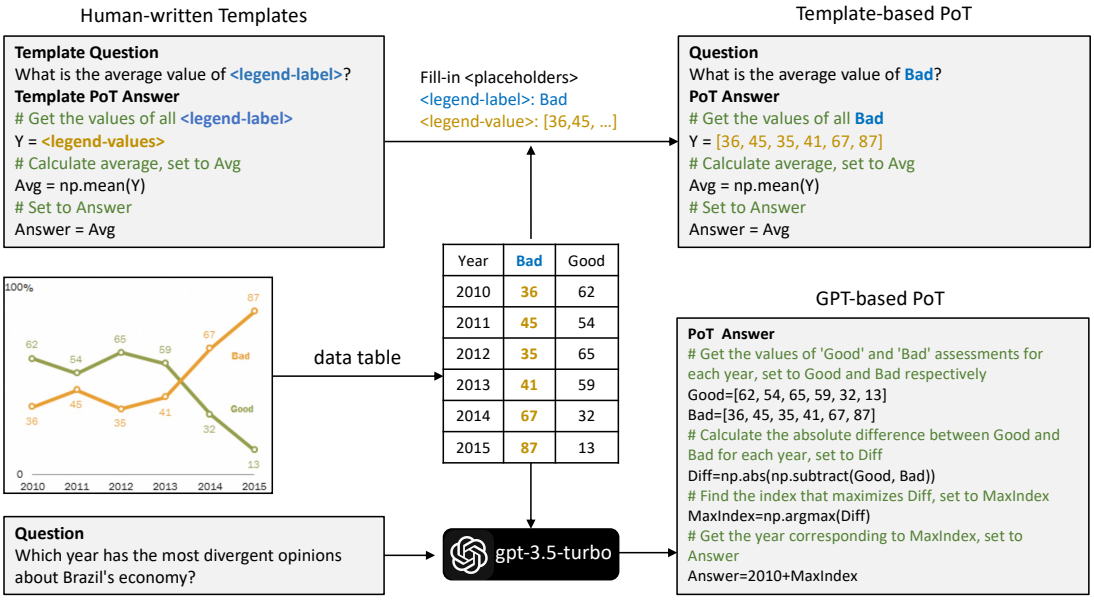


Figure 4: The demonstration of constructing Template-based PoT (upper half) and GPT-based PoT (lower half) in the ChartQA-PoT dataset.

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 (Methani et al., 2020) 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 (van der Walt et al., 2011) 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. We then apply rule-based strategies to remove non-executable PoT answers and unreasonable fill-ins, and finally successfully construct 119,281 (question, PoT pairs) over 17,498 images from ChartQA.

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` (OpenAI, 2023a) 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 the 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. We mix data in each task together jointly for training and distinguish them with task-specified instructions. Note that in ablation studies, we train solely with benchmark datasets due to limited computational resources. The benchmark datasets consist of basic chart understanding evaluations including QA, summary, and chart-to-table generation. We present the detailed composi-

tion of our training data in Appendix A.3.

4 Experiment

4.1 Evaluation Benchmarks

ChartQA provides a short answer to each question based on the chart content (Masry et al., 2022). We report the relaxed accuracy that allows numerical error within 5% as the metric following Masry et al. (2022); Han et al. (2023); Meng et al. (2024). Note that TinyChart with PoT learning can perform ChartQA in the following five settings:
1) **Direct**: model produces short answers directly.
2) **PoT**: model produces Python code and gets final answers through a Python interpreter.
3) **Combine**: we let the model produce PoT answers for calculative questions, and Direct answers for others. Questions contain one of the keywords[1] are considered calculative questions.
4) **Auto**: we train the model to automatically decide whether to produce PoT answers or Direct answers for each question.
5) **Oracle**: we let the model produce both Direct and PoT answers for each question separately, and always choose the correct one.
We evaluate TinyChart under the **Combine** setting by default.
Chart-to-Text aims to generate a summary text based on chart content. We evaluate the model with the Pew dataset (Kantharaj et al., 2022b), and report BLEU4 (Papineni et al., 2002) as the metric[2].
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 (Masry et al., 2022) following (Han et al., 2023; Meng et al., 2024). We report RMS$_{F1}$ (Liu et al., 2023a) as the metric.
OpenCQA Different from ChartQA, OpenCQA evaluates the ability of models to generate free-form answers to the chart-related questions (Kantharaj et al., 2022a). We report BLEU4 as the metric following (Masry et al., 2024).
ChartX is a chart comprehension benchmark that contains more visual types (Xia et al., 2024). We evaluate the ChartX cognition tasks since they are more challenging. It covers Question Answering, Chart Description, Chart Summary, and Chart Redrawing. We report the GPT-Acc for QA and GPT-score for the remaining 3 tasks (Xia et al., 2024).

[1] 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
[2] We calculate BLEU4 with sacrebleu==2.4.1

4.2 Main Results

Table 1 shows an extensive comparison between TinyChart and existing 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 excelling in inference throughput. Specifically, TinyChart@768 achieves an accuracy of 83.60% on ChartQA (Masry et al., 2022), surpassing several closed-source models including GPT-4V (OpenAI, 2023b), Gemini-Ultra (Team et al., 2023), and Qwen-VL-Max (Bai et al., 2023). It also outperforms the previous open-source SOTA model ChartAst (Meng et al., 2024).

We find that previous models performed poorly on the ChartQA-human compared to ChartQA-Aug, with none of them achieving over 70%. This is because the questions posed by human annotators involve more computational problems (Masry et al., 2022) and are more challenging. By leveraging the PoT learning, TinyChart achieves 73.34% on ChartQA-human, which is an improvement of 7.44% over ChartAst (Meng et al., 2024). This demonstrates the effectiveness of the proposed method based on the Program-of-Thoughts.

We observed that models with higher input resolutions generally perform better on chart understanding tasks. However, encoding high-resolution charts leads to a decrease in inference speed. By leveraging visual token merging, TinyChart can accept higher-resolution inputs with a limited increase in computing, thus achieving better performance. Due to the smaller model size and the efficient visual token merging, TinyChart achieves significantly larger inference throughput compared to previous models. It demonstrates that TinyChart can achieve efficient chart understanding with enhanced performance and faster inference.
ChartQA performance in each setting. Table 2 shows the performance comparison under different settings. Note that the performance of ChartAst under the Combine setting is from Meng et al. (2024), which leverages a combination of Direct answers and executive JSON answers. 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 straightforward, the

Table 1: Chart understanding evaluation. Inference throughput is tested on ChartQA using a V100 32G GPU.

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

Table 2: ChartQA performance in different settings.

Model	ChartQA Settings				
	Direct	PoT	Combine	Auto	Oracle
ChartLlama	69.66	-	-	-	-
ChartAst	75.10	-	79.90	-	-
TinyChart@768	**76.36**	**80.84**	**83.60**	**83.48**	**89.12**

Table 3: ChartQA performance on Calculative (Cal.) and Non-calculative (Non-cal.) questions.

Model	Setting	Cal.	Non-cal.	Total
TinyChart@768	Direct	56.64	84.99	76.36
TinyChart@768	PoT	78.98	81.66	80.84
TinyChart@768	Combine	**80.42**	84.99	**83.60**
TinyChart@768	Auto	79.63	**85.16**	83.48

performance under both the Combine setting and Auto setting falls behind the Oracle setting a lot. Further study can be conducted to combine the two answers better.

Calculative and non-calculative questions. We divide the questions in the ChartQA test into two categories: calculative questions (761 of 2500) and non-calculative questions (1739 of 2500) by checking whether they contain any calculative keywords mentioned above. From Table 3, 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). By combining Direct and PoT answers, both Combine setting and Auto setting make further improvements.

Evaluation on ChartX. To further assess the generalizability of TinyChart, We evaluate TinyChart on ChartX-Cognition (Xia et al., 2024) since it covers visually diverse chart types. Note that we do not perform additional fine-tuning in this evaluation. As shown in Table 4, benefiting from PoT learning, TinyChart achieves a 33.35 GPT-Acc on the QA task, even surpassing GPT-4V. Though it falls behind GPT-4V in the other three tasks, TinyChart still outperforms Open-source Chart MLLMs including ChartLlama and ChartAst. It indicates that TinyChart has a strong capability to generalize across various chart types.

Table 4: Evaluation results on ChartX-Cognition.

Model	QA	Summ.	Desc.	Redraw
General MLLM				
Llava1.5	17.19	1.48	1.29	0.75
GPT-4V	33.04	**3.17**	**3.12**	**2.63**
Chart MLLM				
ChartLlama	13.80	1.04	1.02	0.94
ChartAst	30.99	0.33	1.03	0.82
TinyChart@768	**33.35**	1.53	1.64	1.89

4.3 Ablation Studies

We further conduct extensive ablation studies on PoT learning and visual token merging in Table 5.

Ablation on PoT Learning. The upper block in Table 5 shows the model performance with and without training on the PoT data. Comparing Row 2 with 1, we observe that training with template-based PoT improves the accuracy of direct answers (71.12 vs. 70.72). It indicates that PoT learning

Table 5: Ablation study. We train the models only using benchmark datasets in this experiment.

Row	Resolution	GPT PoT	Temp. PoT	Token Merge	Visual Length	Inference Throughput	ChartQA			Chart2Text	Chart2Table
							Direct	PoT	Combine	BLEU4	RMS_{F1}
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

enhances the model's reasoning abilities. At this point, the PoT answers produced by the model are worse than direct answers (55.44 vs. 71.12), which may be due to the inability of the templates to cover all questions. However, when we produce PoT answers for calculative questions and combine them with direct answers, the result outperforms solely direct answers (73.00 vs. 71.12), indicating the advantage of PoT in solving 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 suggest that PoT learning not only strengthens calculations but also enhances reasoning capability.

Ablation on Visual Token Merging. The middle block in Table 5 compares the performance with and without visual token merging under resolution 512×512, and with different numbers of tokens to merge in each layer. Comparing Row 4 and 3, increasing resolution from 384 to 512 brings significant improvements on all 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), since lengthy visual features from standard ViT bring considerable computational expenses for the LLM to process. By adopting the visual token merging, we can control the length of the visual feature by regulating the number of tokens to merge at each layer, 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, 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 5. 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 performance benefits on ChartQA (81.04 vs. 80.76) and Chart-to-Table (88.90 vs. 87.81). It illustrates that by utilizing visual token merging, we can leverage higher-resolution chart images under constrained resources, thereby improving performance.

4.4 Visualization

Visual Token Merging. To investigate the effects of visual token merging, we visualized the token merging results at the final layer of the vision encoder. In Figure 5, we visualize the top 10 groups with the largest numbers of tokens. Each group is outlined with a different color. It reveals that the merged groups typically correspond to blank or colored areas. By compressing these areas down to a single token, our visual token merging module can thus reduce the length of the encoded sequence without losing much information, thereby achieving efficient visual encoding.

Case Study. In Figure 6, we present a case study on ChartQA. As shown in Figure 6(a), 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, our TinyChart can accept higher-resolution inputs and thus successfully

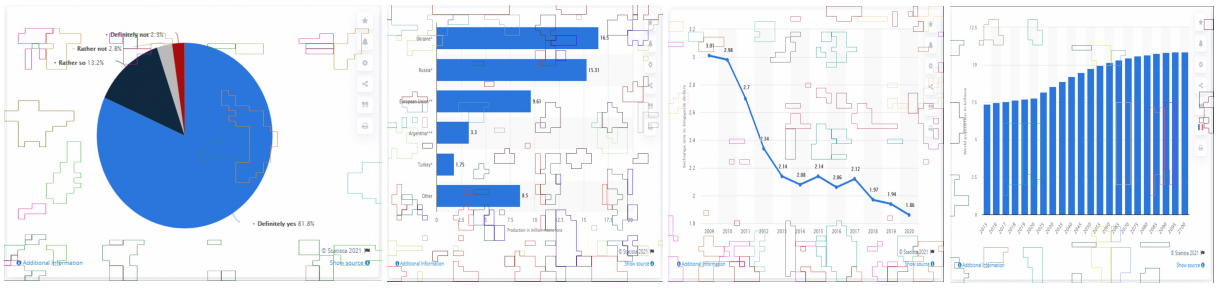


Figure 5: Token merging visualization. Top 10 groups with the most merged tokens are outlined in different colors.

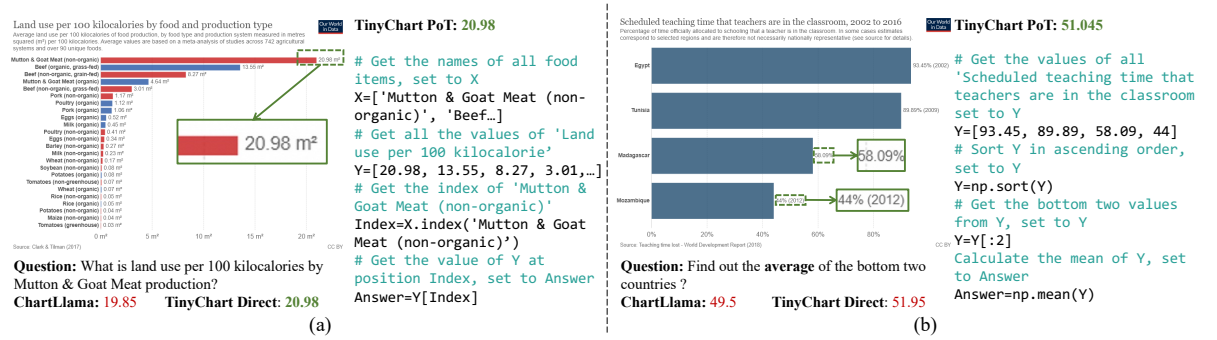


Figure 6: Case studies on ChartQA. We compare TinyChart@768 with ChartLlama.

find clues related to the questions. Meanwhile, ChartLlama suffers from numerical errors when faced with calculative questions in Figure 6 (b), and TinyChart can provide a PoT answer that accurately solves the problem. More case analyses is presented in Appendix C.

5 Conclusion

This paper introduces TinyChart, a chart understanding MLLM with 3 billion parameters. To address the inefficiency of lengthy visual token sequences with high-resolution images, TinyChart injects the visual token merging module that merges similar vision tokens, 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. TinyChart achieves SOTA performance on multiple chart understanding benchmarks with fewer parameters and larger inference throughput. Extensive ablation studies confirm the effectiveness of our methods.

6 Limitations

While TinyChart demonstrates notable performance and efficiency, it is still susceptible to hallucination issues in summary generation that are commonly associated with MLLMs, and therefore it may generate misleading information about chart content. Further research can be conducted to mitigate this hallucination problem in chart understanding. Meanwhile, we observed that TinyChart encounters challenges in estimating the value of a data point without surrounding OCR words, even with increased input resolution. Potential future work could focus on developing methods to improve the accuracy of these non-OCR estimations.

7 Acknowledgements

This work was partially supported by the Beijing Natural Science Foundation (No. L233008) and the National Natural Science Foundation of China (No. 62072462).

References

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. *Preprint*, arXiv:2308.12966.

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*.

Jinyue Chen, Lingyu Kong, Haoran Wei, Chenglong Liu, Zheng Ge, Liang Zhao, Jianjian Sun, Chunrui Han, and Xiangyu Zhang. 2024a. Onechart: Purify the chart structural extraction via one auxiliary token. *Preprint*, arXiv:2404.09987.

Wenhu 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*.

Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, Ji Ma, Jiaqi Wang, Xiaoyi Dong, Hang Yan, Hewei Guo, Conghui He, Botian Shi, Zhenjiang Jin, Chao Xu, Bin Wang, Xingjian Wei, Wei Li, Wenjian Zhang, Bo Zhang, Pinlong Cai, Licheng Wen, Xiangchao Yan, Min Dou, Lewei Lu, Xizhou Zhu, Tong Lu, Dahua Lin, Yu Qiao, Jifeng Dai, and Wenhai Wang. 2024b. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *Preprint*, arXiv:2404.16821.

Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Bin Wang, Linke Ouyang, Xilin Wei, Songyang Zhang, Haodong Duan, Maosong Cao, et al. 2024a. Internlm-xcomposer2: Mastering free-form text-image composition and comprehension in vision-language large model. *arXiv preprint arXiv:2401.16420*.

Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Bin Wang, Linke Ouyang, Songyang Zhang, Haodong Duan, Wenwei Zhang, Yining Li, et al. 2024b. Internlm-xcomposer2-4khd: A pioneering large vision-language model handling resolutions from 336 pixels to 4k hd. *arXiv preprint arXiv:2404.06512*.

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.

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*.

Jiayun Fu, Bin B Zhu, Haidong Zhang, Yayi 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*, pages 2786–2795.

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*.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging non-linearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

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*.

Anwen Hu, Shizhe Chen, and Qin Jin. 2021. Question-controlled text-aware image captioning. In *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21, page 3097–3105, New York, NY, USA. Association for Computing Machinery.

Anwen Hu, Yaya Shi, Haiyang Xu, Jiabo Ye, Qinghao Ye, Ming Yan, Chenliang Li, Qi Qian, Ji Zhang, and Fei Huang. 2024a. mplug-paperowl: Scientific diagram analysis with the multimodal large language model. *Preprint*, arXiv:2311.18248.

Anwen Hu, Haiyang Xu, Jiabo Ye, Ming Yan, Liang Zhang, Bo Zhang, Chen Li, Ji Zhang, Qin Jin, Fei Huang, and Jingren Zhou. 2024b. mplug-docowl 1.5: Unified structure learning for ocr-free document understanding. *Preprint*, arXiv:2403.12895.

Kung-Hsiang Huang, Hou Pong Chan, Yi R. Fung, Haoyi Qiu, Mingyang Zhou, Shafiq Joty, Shih-Fu Chang, and Heng Ji. 2024a. From pixels to insights: A survey on automatic chart understanding in the era of large foundation models. *Preprint*, arXiv:2403.12027.

Qidong Huang, Xiaoyi Dong, Pan Zhang, Bin Wang, Conghui He, Jiaqi Wang, Dahua Lin, Weiming Zhang, and Nenghai Yu. 2024b. Opera: Alleviating hallucination in multi-modal large language models via over-trust penalty and retrospection-allocation. *Preprint*, arXiv:2311.17911.

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. *Preprint*, arXiv:2312.06968.

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*, pages 5648–5656.

Shankar Kantharaj, Xuan Long Do, Rixie Tiffany Leong, Jia Qing Tan, Enamul Hoque, and Shafiq Joty. 2022a. OpenCQA: Open-ended question answering with charts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11817–11837, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Shankar Kantharaj, Rixie Tiffany Leong, Xiang Lin, Ahmed Masry, Megh Thakkar, Enamul Hoque, and Shafiq Joty. 2022b. 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)*, pages 4005–4023, Dublin, Ireland. Association for Computational Linguistics.

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. *Preprint*, arXiv:2311.16922.

Bo Li, Kaichen Zhang, Hao Zhang, Dong Guo, Renrui Zhang, Feng Li, Yuanhan Zhang, Ziwei Liu, and Chunyuan Li. 2024. Llava-next: Stronger llms supercharge multimodal capabilities in the wild.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. 2023a. Evaluating object hallucination in large vision-language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 292–305.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023b. Textbooks are all you need ii: phi-1.5 technical report. *Preprint*, arXiv:2309.05463.

Ziyi Lin, Chris Liu, Renrui Zhang, Peng Gao, Longtian Qiu, Han Xiao, Han Qiu, Chen Lin, Wenqi Shao, Keqin Chen, Jiaming Han, Siyuan Huang, Yichi Zhang, Xuming He, Hongsheng Li, and Yu Qiao. 2023. Sphinx: The joint mixing of weights, tasks, and visual embeddings for multi-modal large language models. *Preprint*, arXiv:2311.07575.

Fangyu Liu, Julian Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. 2023a. DePlot: One-shot visual language reasoning by plot-to-table translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10381–10399, Toronto, Canada. Association for Computational Linguistics.

Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Eisenschlos. 2023b. 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)*, pages 12756–12770, Toronto, Canada. Association for Computational Linguistics.

Fuxiao Liu, Xiaoyang Wang, Wenlin Yao, Jianshu Chen, Kaiqiang Song, Sangwoo Cho, Yaser Yacoob, and Dong Yu. 2023c. Mmc: Advancing multimodal chart understanding with large-scale instruction tuning. *arXiv preprint arXiv:2311.10774*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023d. Improved baselines with visual instruction tuning. *Preprint*, arXiv:2310.03744.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024a. Visual instruction tuning. *Advances in neural information processing systems*, 36.

Yuliang Liu, Zhang Li, Biao Yang, Chunyuan Li, Xucheng Yin, Cheng lin Liu, Lianwen Jin, and Xiang Bai. 2024b. On the hidden mystery of ocr in large multimodal models. *Preprint*, arXiv:2305.07895.

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*, pages 2263–2279.

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. *Preprint*, arXiv:2305.14761.

Ahmed Masry, Mehrad Shahmohammadi, Md Rizwan Parvez, Enamul Hoque, and Shafiq Joty. 2024. Chartinstruct: Instruction tuning for chart comprehension and reasoning. *Preprint*, arXiv:2403.09028.

Fanqing Meng, Wenqi Shao, Quanfeng Lu, Peng Gao, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024. Chartassisstant: A universal chart multimodal language model via chart-to-table pre-training and multitask instruction tuning. *arXiv preprint arXiv:2401.02384*.

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*, pages 1527–1536.

Jason Obeid and Enamul Hoque. 2020. Chart-to-text: Generating natural language descriptions for charts by adapting the transformer model. *CoRR*, abs/2010.09142.

OpenAI. 2023a. Gpt-3.5-turbo. `https://platform.openai.com/docs/models/gpt-3-5-turbo`.

OpenAI. 2023b. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

OpenAI. 2024. Hello gpt-4o. `https://openai.com/index/hello-gpt-4o/`.

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*, 35:27730–27744.

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*, pages 311–318.

Raian Rahman, Rizvi Hasan, Abdullah Al Farhad, Md. Tahmid Rahman Laskar, Md. Hamjajul 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*. Https://caiac.pubpub.org/pub/ujhjycsw.

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*, pages 4035–4045.

Amanpreet 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)*.

Benny Tang, Angie Boggust, and Arvind Satyanarayan. 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)*, pages 7268–7298, Toronto, Canada. Association for Computational Linguistics.

Gemini Team, Rohan Anil, Sebastian 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*.

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):22–30.

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.

Junyang Wang, Yuhang Wang, Guohai Xu, Jing Zhang, Yukai Gu, Haitao Jia, Ming Yan, Ji Zhang, and Jitao Sang. 2023a. An llm-free multi-dimensional benchmark for mllms hallucination evaluation. *arXiv preprint arXiv:2311.07397*.

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. 2023b. Evaluation and analysis of hallucination in large vision-language models. *Preprint*, arXiv:2308.15126.

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. *Preprint*, arXiv:2402.12185.

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. 2023a. Ureader: Universal ocr-free visually-situated language understanding with multimodal large language model. In *EMNLP (Findings)*, pages 2841–2858. Association for Computational Linguistics.

Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, Chenliang Li, Yuanhong Xu, Hehong Chen, Junfeng Tian, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. 2024. mplug-owl: Modularization empowers large language models with multimodality. *Preprint*, arXiv:2304.14178.

Qinghao Ye, Haiyang Xu, Ming Yan, Chenlin Zhao, Junyang Wang, Xiaoshan Yang, Ji Zhang, Fei Huang, Jitao Sang, and Changsheng Xu. 2023b. mplug-octopus: The versatile assistant empowered by a modularized end-to-end multimodal llm. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9365–9367.

Qinghao Ye, Haiyang Xu, Jiabo Ye, Ming Yan, Anwen Hu, Haowei Liu, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. 2023c. mplug-owl2: Revolutionizing multi-modal large language model with modality collaboration. *Preprint*, arXiv:2311.04257.

Zihao Yue, Liang Zhang, and Qin Jin. 2024. Less is more: Mitigating multimodal hallucination from an eos decision perspective. *arXiv preprint arXiv:2402.14545*.

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*, pages 11975–11986.

Liang Zhang, Anwen Hu, Jing Zhang, Shuo Hu, and Qin Jin. 2023a. Mpmqa: multimodal question answering on product manuals. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13958–13966.

Pan Zhang, Xiaoyi Dong Bin Wang, Yuhang Cao, Chao Xu, Linke Ouyang, Zhiyuan Zhao, Shuangrui Ding, Songyang Zhang, Haodong Duan, Hang Yan, et al. 2023b. Internlm-xcomposer: A vision-language large model for advanced text-image comprehension and composition. *arXiv preprint arXiv:2309.15112*.

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, pages 1413–1422.

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. *Preprint*, arXiv:2402.14289.

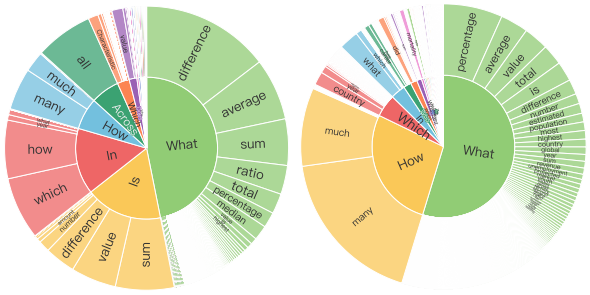
A Implementation Details

A.1 ChartQA-PoT Statistic

We build ChartQA-PoT based on the images and questions in the training split of ChartQA (Masry et al., 2022). ChartQA-PoT consists of two subsets: Template-based PoT and GPT-based PoT. We present the statistics over ChartQA-PoT in Table 6. We find that answers provided by `gpt-3.5-turbo` are longer than template-based PoT, since they cover more diverse scenarios.

Table 6: 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



(a) Template PoT. (b) GPT PoT.

Figure 7: First 2-gram of the questions in ChartQA-PoT after removing stop words.

Table 7: ChartQA performance after further fine-tuning.

Model	Training Data	Direct	PoT
TinyChart@768	-	76.36	80.84
TinyChart@768	GPT-4o-PoT	**77.68**	**81.12**

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 7. It is observed that GPT-PoT covers more diverse questions 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 8 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.

A.3 Training details

TinyChart is initialized from TinyLlava (Zhou et al., 2024), which utilizes the SigLIP (Zhai et al., 2023) as the vision encoder and Phi-2 (Li et al., 2023b) 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. We present the detailed composition of our training data and their license in Table 8.

B Further improve with GPT-4o

We further apply the more recent MLLM GPT-4o (OpenAI, 2024) to generate the PoT answers. Instead of using data tables, we provide the chart images directly to GPT-4o. The constructed GPT-4o-PoT dataset contains 23,437 QAs over 16,474 charts after removing incorrect answers, demonstrating higher accuracy than GPT-3.5-Turbo, which resulted in 21,303 QAs over 15,521 charts. We finetune our TinyChart@768 on GPT4o-PoT. The results in Table 7 show that learning with GPT4o-generated PoT leads to further improvements. This demonstrates that our PoT learning strategy is compatible with stronger data generators.

C Case study

We conduct more case studies with TinyChart when conducting chart question answering, chart-

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>`[a]. 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:

[a]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 8: Instructions used for generating GPT-based PoT.

to-table, chart-to-text, and chart redrawing in Figure 9, 10, 11, and 12.

Chart Question Answering. In Figure 9, As shown in Figure 9 (a-c), compared to ChartLlama, TinyChart captures visual details for problem solving due to higher input resolution. Also, our Program-of-Thoughts learning strategy enables TinyChart to produce Python codes for numerical calculations and successfully avoids errors in computation (Figure 9(d-e)). 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 10 (a-c), thanks to its excellent text recognition ability with high-resolution input. However, as shown in Figure 10 (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

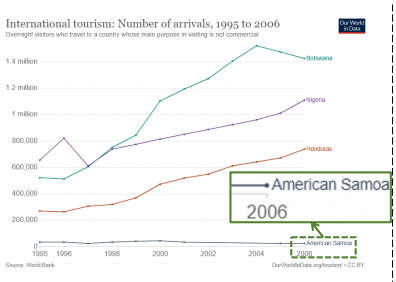
Table 8: Datasets used for training TinyChart. The benchmark datasets consist of basic chart understanding evaluations including QA, summary, and chart-to-table generation. Note that in ablation studies, we only use the benchmark datasets for training due to limited computational resources.

Dataset	License	Benchmark	Samples
Chart question answer			
ChartQA (Masry et al., 2022)	GPL-3.0	✓	28,299
ChartQA-PoT	-	✓	140,584
PlotQA (Methani et al., 2020)	CC-BY-4.0		157,070
DVQA (Kafle et al., 2018)	Tencent		200,000
OpenCQA (Kantharaj et al., 2022a)	GPL-3.0		5,407
Chart-to-text generation			
Pew (Kantharaj et al., 2022b)	GPL-3.0	✓	7,892
Statista (Kantharaj et al., 2022b)	GPL-3.0	✓	29,589
OpenCQA (Kantharaj et al., 2022a)	GPL-3.0		5,407
Vistext (Tang et al., 2023)	GPL-3.0		11,171
ChartSumm (Rahman et al., 2023)	-		75,255
Chart2Text-8k (Obeid and Hoque, 2020)	-		7,862
Chart-to-table generation			
ChartQA (Masry et al., 2022)	GPL-3.0	✓	19,373
PlotQA (Methani et al., 2020)	CC-BY-4.0		190,720
Chart2Text-8k (Obeid and Hoque, 2020)	-		8,305
DVQA (Kafle et al., 2018)	Tencent		300,000
Statista (Kantharaj et al., 2022b)	GPL-3.0		29,589
Chart instruction following			
ChartLlama (Han et al., 2023)	MIT		148,398
Total			**1,364,921**

lacks the ability to understand spatial relationships across large areas.

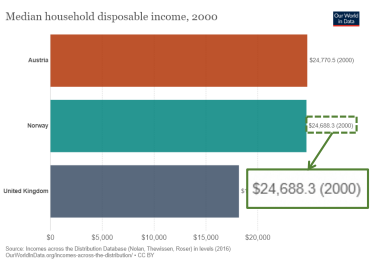
Chart-to-Text. From Figure 11, we observe that the model can understand the data presented in the chart and generate descriptions and summaries in natural language. Though it can retrieve the data values correctly, we find it sometimes produces contents that do match the chart as shown in Figure 11 (c-d). This may be due to the inherent limitations of hallucination in MLLMs (Rohrbach et al., 2018; Li et al., 2023a; Wang et al., 2023b,a), and may be alleviated by addressing hallucinations (Leng et al., 2023; Huang et al., 2024b; Jiang et al., 2024; Yue et al., 2024).

Chart redrawing. We present four cases of chart redrawing in Figure 12. As shown in Figure 12 (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 12 (d)). This may be mitigated by improving the coverage of different chart types in training data through automatic data construction techniques (Han et al., 2023; Xia et al., 2024).



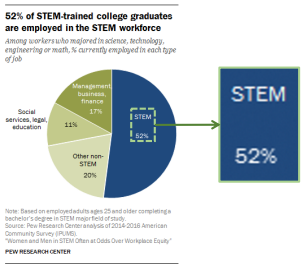
Question: How many countries are represented in the chart?
ChartLlama: 3 **TinyChart Direct**: 4

(a)



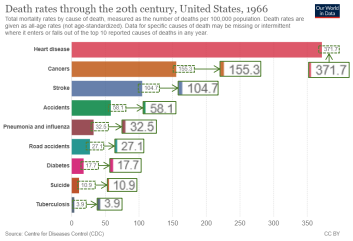
Question: What is the value of middle bar in the graph?
ChartLlama: 19178.33 **TinyChart Direct**: 24688.3

(b)



Question: Is the percentage value of "STEM" segment 52?
ChartLlama: No **TinyChart Direct**: Yes

(c)

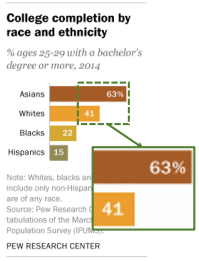


Question: Is the **median** value of all the bars **greater than** 30?
ChartLlama: No **TinyChart Direct**: Yes

(d)

TinyChart PoT: Yes

```
# Get the values of all
'Death rates through the
20th century', set to Y
Y=[371.7, 155.3, 104.7,
58.1, 32.5, 27.1, 17.7,
10.9, 3.9]
# Calculate the median of Y,
set to Median
Median=np.median(Y)
# Check if Median is
greater than 30, set to
Answer
Answer=np.greater(Median,30)
```



Question: What is the **difference** between Asians and Whites degree distribution?
ChartLlama: 21 **TinyChart Direct**: 22

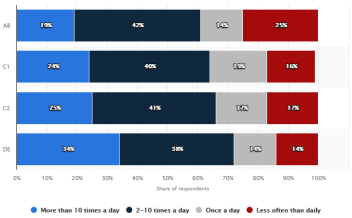
(e)

TinyChart PoT: 22

```
# Get the values of all
'Values' in the chart, set
to Y
Y=[15, 22, 41, 63]
# Get the difference
between the value of Asians
and Whites, set to Subtract
Subtract=np.subtract(Y[3],
Y[2])
# Calculate the absolute
value of Subtract, set to
Answer
Answer=np.abs(Subtract)
```

Figure 9: More cases on ChartQA. We compare TinyChart@768 with ChartLlama.

Input: Generate underlying data table for the chart.

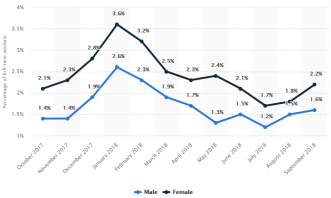


(a)

TinyChart:

Characteristic | More than 10 times a day | 2-10 times a day | Once a day | Less often than daily
AB | 19% | 42% | 14% | 25%
C1 | 24% | 40% | 19% | 16%
C2 | 25% | 41% | 17% | 17%
DE | 34% | 38% | 14% | 14%

Input: Generate underlying data table for the chart.

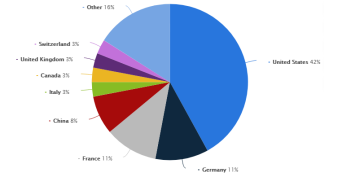


(b)

TinyChart:

Characteristic | Male | Female
September 2018 | 1.6% | 2.2%
August 2018 | 1.5% | 1.8%
July 2018 | 1.2% | 1.7%
June 2018 | 1.5% | 2.1%
May 2018 | 1.3% | 2.4%
April 2018 | 1.7% | 2.3%
March 2018 | 1.9% | 2.5%
February 2018 | 2.3% | 3.2%
January 2018 | 2.6% | 3.6%
December 2017 | 1.9% | 2.8%
November 2017 | 1.4% | 2.3%
October 2017 | 1.4% | 2.1%

Input: Generate underlying data table for the chart.

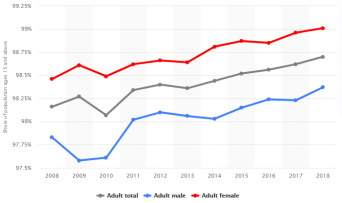


(c)

TinyChart:

Characteristic | Retail sales share
United States | 42%
Germany | 11%
France | 11%
China | 8%
Italy | 3%
Canada | 3%
United Kingdom | 3%
Switzerland | 3%
Other | 16%

Input: Generate underlying data table for the chart.

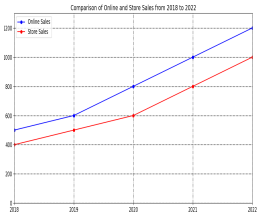


(d)

TinyChart:

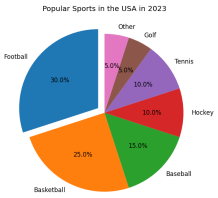
Characteristic | Adult total | Adult male | Adult female
2008 | 98.29% | 97.83% | 98.51%
2009 | 98.36% | 97.67% | 98.62%
2010 | 98.28% | 97.65% | 98.56%
2011 | 98.44% | 98.07% | 98.70%
2012 | 98.47% | 98.10% | 98.74%
2013 | 98.43% | 98.06% | 98.70%
2014 | 98.49% | 98.03% | 98.81%
2015 | 98.54% | 98.08% | 98.87%
2016 | 98.59% | 98.17% | 98.86%
2017 | 98.65% | 98.21% | 98.93%
2018 | 98.70% | 98.32% | 98.98%

Figure 10: Examples of chart-to-table extraction of TinyChart@768. Wrong values are marked red.



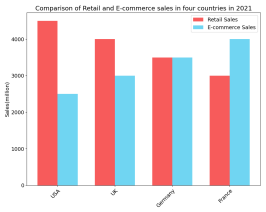
Input: Create a brief summarization or extract key insights based on the chart image.
TinyChart: The chart illustrates the sales trends for online and store sales from 2018 to 2022. It shows that online sales steadily increased from 500 in 2018 to 1200 in 2022, while store sales also experienced an increase from 400 in 2018 to 1000 in 2022. The data indicates that while online sales growth has been consistently positive, store sales growth has been relatively slower. This could suggest a shift in consumer preference or a need for a more extensive retail network.

(a)



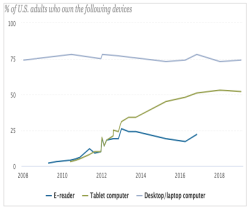
Input: Create a brief summarization or extract key insights based on the chart image.
TinyChart: The chart presents the popularity of various sports in the United States in 2023. Football is the most popular sport with a 30% popularity rating, followed by Basketball at 25%. Baseball and Hockey have a 15% and 10% popularity respectively. Meanwhile, Golf and Tennis have a 5% and 10% popularity rating respectively. The least popular sport is "Other" with a 5% rating.

(b)



Input: Create a brief summarization or extract key insights based on the chart image.
TinyChart: The chart illustrates the retail and e-commerce sales figures in four major online markets: USA, UK, Germany, and France in 2021. Retail sales dominated in the USA with a value of 4500, followed by the UK with 4000. Germany and France saw 3500 and 3000 retail sales respectively, while e-commerce sales were the highest in France with 4000, and second highest in the USA with 2500.

(c)

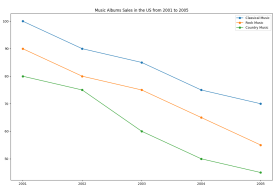


Input: Write a summary for this chart.
TinyChart: The share of Americans who own desktop or laptop computers has remained relatively stable in recent years, while the share who own tablet computers and e-readers has grown. Today, about three-quarters of U.S. adults (74%) own a desktop or laptop computer, up from 70% in early 2014.

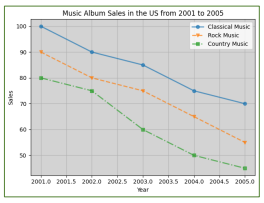
(d)

Figure 11: Cases of chart-to-text generation by TinyChart@768. Correct contents are shown in green and wrong contents are marked red.

Input: Redraw the chart with Python code.

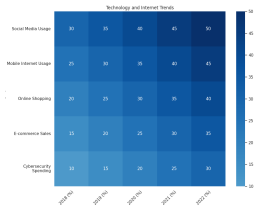


TinyChart:

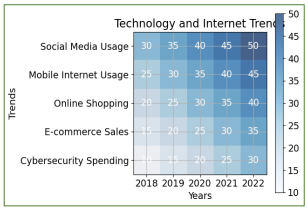


(a)

Input: Redraw the chart with Python code.

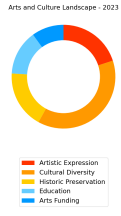


TinyChart:

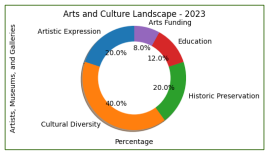


(b)

Input: Redraw the chart with Python code.

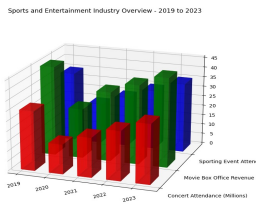


TinyChart:

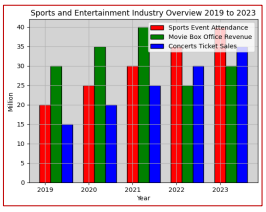


(c)

Input: Redraw the chart with Python code.



TinyChart:



(d)

Figure 12: 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.