$\gamma GAMMAS$: Improving Mathematical Reasoning in Vision Language Models Through Synthetic Data Generation

Ramgopal Venkateswaran, Shubhra Mishra
Stanford University
Stanford, CA, 94305
ram1998@stanford.edu, shubhra@stanford.edu

Abstract

Vision Language Models (VLMs) have seen a recent exponential growth in their capabilities. However, they still face significant gaps, specifically as it relates to complex tasks like mathematical reasoning. In this work, we present GAMMAS: a pipeline that uses GPT-4 and code-generation to synthesize a fine-tuning and evaluation dataset containing 860 bar and line charts, and relevant mathematical questions. We then use this dataset to fine-tune the InternLM-XComposer2-VL-1.8B model. We show a 6.5% and 12.5% increase in performance on the bar and line chart tasks in the MathVista benchmark without a loss in overall performance. We also note that performance on other categories of tasks in MathVista increases as well.

1. Introduction

Large language models (LLMs) have shown reasonable progress in their math problem-solving abilities over the last few years. Vision language models (VLMs) are a class of multimodal models that combine text input with image data to produce either text or a combination of image and text as the output - such models have attracted a lot of recent attention (e.g. [2] and [27]). Such models have only very recently reached a level where they can be expected to tackle visual reasoning problems. The MathVista dataset benchmarks these capabilities, conducting finegrained analyses based on the specific skills and knowledge domains tested by the problems in the dataset [19]. While leading VLMs like GPT-4V and Gemini Pro outdo humans in some visual problem solving contexts, considerable gaps still exist. In this work, we aim to address these gaps via the following contributions:

1. We propose GAMMAS: a pipeline to Generate Advanced Multi-modal Mathematical And Synthetic data. Overall, we generate 860 training samples and a validation set of size 200, comprising questions that test visual reasoning using bar charts and line plots.
2. We finetune InternLM-XComposer2-VL-1.8B [6], a small but performant VLM using our generated training data, and evaluate its performance both with respect to our test data as well as on the MathVista "testmini" dataset. Within MathVista, we investigate the performance improvement on not just line plot and bar chart based questions, but also on other related categories to understand the transfer learning capabilities of the model.

Specifically, the input to our finetuned model comprises a mathematical reasoning question and an accompanying figure. We restrict our attention to multiple choice questions (MCQs), get a response from either base or fine-tuned model, extract the final answer from the response, and evaluate it against the correct answer to look at its accuracy in terms of exact match (i.e. is the selected option the correct one).

2. Related Work

Synthetic Data Generation A lot of work has explored generating synthetic data in order to train/finetune models. Gunasekar et. al. showed the importance of data quality during training by using GPT-3.5 to generate textbook-quality synthetic data to train a model for code-generation [10]. In TinyStories, Eldan et. al. train a small language model to write coherent and creative short stories, only using synthetic data generated by GPT-3.5 [8]. Similarly, in TinyGSM, Liu et. al. use GPT-3.5 to augment the GSM8K dataset to train a small language model with significant mathematical capabilities [17].

Mathematical Reasoning in Language Models Some of the early natural-language math problem solving benchmarks included MAWPS in [16] and work in [25]. The most widely adopted benchmarks for math word problem solving are the GSM8K and MATH datasets [5, 11]. Since 2021,

model performance has generally become saturated on these benchmarks, as a result of both model improvement and data contamination. Using LLMs for mathematical reasoning in the formal direction has also been explored, with a lot of work being done in autoformalization and automated theorem proving, although results show that both tasks are extremely challenging, and we have a long way to go [26, 14].

Vision Models Like LLMs, vision models have also only recently become very powerful, letting them succeed at tasks beyond simple classification and caption-generation [24]. Object counting is another task that vision models have been evaluated on [21]. Datasets like `ChartQA` and `FigureQA` are examples of benchmarks first used to measure vision models' abilities for scientific tasks [20, 15]. The `MathVista` benchmarks combines 20+ such benchmarks to create a comprehensive evaluation of vision models' visual math problem solving abilities [19].

InternLM InternLM-XComposer2-VL [7] is a recent vision language model that is of particular interest since the 7B version is currently the fifth highest ranked vision language model on the MathVista public test leaderboard and competitive with models of much larger size. We focus on the 1.8B version, which is more feasible to tune with fewer compute resources.

InternLM-XComposer2-VL is composed of a language model (InternLM-2 [4]) and a vision encoder (OpenAI ViT-Large model trained using CLIP [23]) that are further trained to be aligned. The image is segmented into a 35x35 grid and embeddings are generated for each part of the image using the vision encoder. There is then a small MLP layer on top of them, and the final image embeddings are then inserted among the text embeddings (generated by the initial embedding layer of the LLM) at the appropriate position where the image should be within the text input. The model is pretrained on such multimodal inputs in a multitask way (for tasks such as captioning, question answering, free-form text image composition, etc.) A unique aspect here is that it is done through partial LoRA, where LoRA is only applied on the image tokens, illustrated in Figure 1 (taken from [7]). The purpose of this is to not disrupt the pre-trained language embeddings too much to preserve information gained during language model pre-training.

3. Methods

Our work is comprised of two parts: generating synthetic data and filtering high-quality samples, as well as finetuning the InternLM models using LoRA and DoRA, which we compare later.

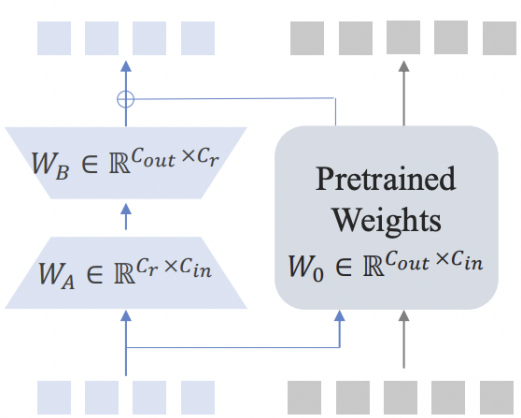

Figure 1. Figure and caption both taken from [4]: The illustration of the Partial-LoRA. The blue tokens represent the visual tokens and the gray tokens are the language tokens. Our Partial-LoRA is only applied to the visual tokens.

3.1. Generating Synthetic Multimodal Data

In this paper, we first propose an automatic pipeline to generate high-quality multimodal synthetic data for visual math reasoning. An example of a human-annotated sample from `MathVista` is shown in Figure 5. Because we have seen the importance of high-quality data in training strong models, we use `GPT-4` to generate synthetic data, as shown in Figure 2. In our prompts, we specify three different things. First, we specify the type of figure to be generated (bar chart vs. line plot). Next, to force diversity, we include information in the prompt that creates linguistic and visual diversity in the data. For example, we prompt `GPT-4` with a specific topic (e.g. Space Exploration, Cities, Dogs, Grades, etc.) and question type (free-form, MCQ, true/false). To ensure that the charts we generate are visually diverse, we prompt `GPT-4` to generate code that creates a certain number of elements, uses specific colors (that we randomly choose), uses specific hashes (that we also randomly generate), etc.

3.2. LLM-as-a-Judge

To improve data quality further, we implement implement LLM-as-a-Judge using `GPT-4` [28]. That is, once we have generated the code, question, and answer, we prompt `GPT-4` to check whether or not the answer correctly solves the question. If it does not, we filter out that sample. By using the code as a proxy for the generated image, we're able to leverage the strengths LLMs have shown [22, 3] to filter out bad-quality multimodal data. We also manually checked the ground-truth value vs the LLM-as-a-judge result for 100 samples for each type of figure, and include the precision and recall analysis in Table 1.

Once we have filtered out bad samples, we have a com-

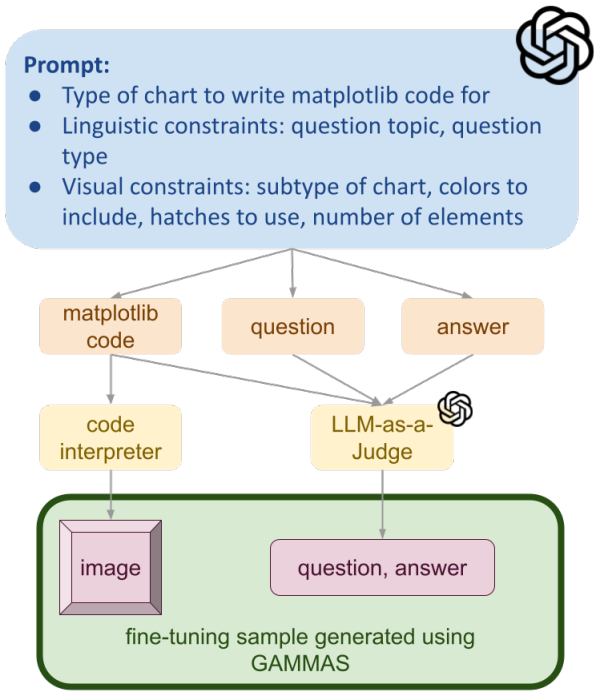

Figure 2. GAMMAS Data Generation Pipeline

Figure Type	Precision (%)	Recall (%)
Bar Chart	100.0	85.2
Line Plot	95.2	93.1

Table 1. Precision and recall analysis of LLM as a judge on our generated data. The precision refers to what percent of the outputs that the judge LLM marked as correct were actually correct based on our ground truth human evaluation, and the recall refers to the percent of all correct answers that were selected as accurate by the LLM.

plete dataset where each problem is associated with a type of chart (bar or line), contains an image, a question, a corresponding answer, and in some cases, answer choices. For cases with answer choices, we noticed a bias in `GPT-4` in that it generated the final answer to be B more often than other choices. To mitigate this, for questions with multiple answer choices, we shuffle the ordering of options as a post-processing step.

3.3. Finetuning

We use LoRA and DoRA, two parameter-efficient finetuning (PEFT) methods in order to efficiently fine-tune the language model component of InternLM-XComposer2-VL-1.8B on a single L4 GPU. We also jointly fine-tune the vision encoder - for this, we do full fine-tuning (because this has relatively a much smaller number of parameters). Note that we do not fine-tune the final MLP (also known as the sampler) that connects the vision encoder and the input of the language model (it converts vision encoder outputs into embeddings that are consumed by the language model).

We now briefly go over how LoRA and DoRA work:

1. Low-Rank Adaptation (LoRA) [13]: Since tuning the many dense layers in a 1.8B transformer model requires additional compute and GPU memory, LoRA fixes the existing weights and instead learns an additional low-rank update matrix for each dense parameter matrix in the model. As mentioned in equation 3 of [13], for a given dense weight $W_0 \in \mathcal{R}^{d\times k}$, we learn matrices $A \in \mathcal{R}^{d\times r}, B \in \mathcal{R}^{r\times k}$, and compute the new forward pass based on $W = W_0 + AB$ instead of W_0. The trick now is that we now don't need to enable the backward pass for the existing W_0 parameters in the model which remain the same, and only need to propagate gradients through the A and B matrices, which we can make much cheaper by choosing $r << d, k$. This saves on GPU memory needed as well as training time. Note that at inference time, we also have no additional costs because we can simply update the weight matrices in our saved model to use $W = W_0 + AB$ everywhere.

2. Weight-Decomposed Low-Rank Adaptation (DoRA) [18]: The concept of DoRA is simply to have an additional parameter to tune the magnitude of the weight matrix instead of just manipulating it by adding a low-rank matrix. It can be formulated as now replacing each weight matrix W_0 with $W = m\frac{W_0+BA}{\|W_0+BA\|_c}$ (as mentioned in equation 5 of [18]). Note that $\|.\|_c$ denotes the column-wise L2 norm of a matrix. The key difference here is that we now have a third learned parameter m that can accordingly manipulate the magnitude of the weight matrices (this is similar in flavor to the learned scaling parameter for batch norm, though of course the norm isn't taken over the batch here, just over columns of the weight matrix). The step-by-step process is also illustrated in 3, taken from [18]. Similar to LoRA, there are no additional inference costs. But a key thing to note is that the memory consumption when training DoRA is notably higher than for LoRA - and the reason for this is that the division by $\|W_0 + BA\|_c$ makes the gradient for W different from the gradient for BA and we need to appropriately handle the division in the computation graph. The authors found that simply treating $\|W_0 + BA\|_c$ as a constant does not result in much less accuracy and reduces much of the additional memory costs (though still more than LoRA) and so they implement this as a workaround to still train DoRA in a memory-efficient manner.

We utilize the InternLM-XComposer library, which provides a Huggingface-based framework for finetuning this

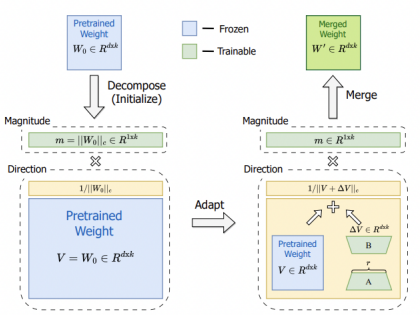

Figure 3. This is figure 1 from [18] and illustrates how DoRA modifies the original weights.

series of models with LoRA - we slightly modify it to also support DoRA.

The overall training set-up aside from the caveats above is standard: the loss function for the model is the cross entropy loss and it is trained on next-token prediction for the generated text (note that we only train it for language-based output, not multimodal output), where the expected output will be the next language token (the tokenizer used throughout is the InternLM-2 tokenizer) and we will take the cross-entropy loss between the outputs of our final softmax layer (which has size $|V|$ where V is the vocabulary of tokens) and the next output token.

4. Dataset and Features

The training set we generate using GAMMAS contains 860 training samples, with 460 bar charts and 400 line plots. We also generate a validation set with 200 samples. Overall, generating this dataset cost under $\$100$, a fraction of what it would cost to create a dataset of this scale using human annotators.

Figure 4 shows an example of a Bar Chart and its related question generated using GAMMAS. Figure 5 shows an example of a Bar Chart and its related question from MathVista. We selected two bar charts with a similar setup to highlight key strengths of our dataset. First, we see visual diversity, because we specifically generate charts with a variety of colors, hatches, and numbers of elements. The chart topics are also more coherent, and the question is notably more challenging than the question in Figure 5.

5. Experiments

Training for all experiments was done on a single L4 GPU. We tuned the following hyperparameters:

1. r in LoRA and DoRA: We tune r, the rank of the low-rank matrix that is added in LoRA and DoRA, trying values ranging from 2 to 32.

2. Learning Rate: We tuned around the value of $5e-5$, ranging from values 4 times below to 4 times above

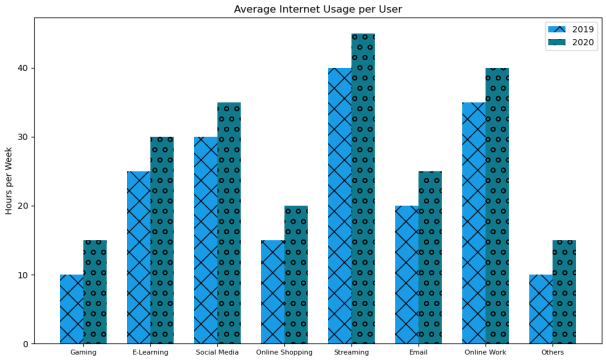

Figure 4. Example Bar Chart Generated Using GAMMAS. Associated question: According to the bar chart, what was the percentage increase in hours spent by users on Online Shopping from 2019 to 2020?

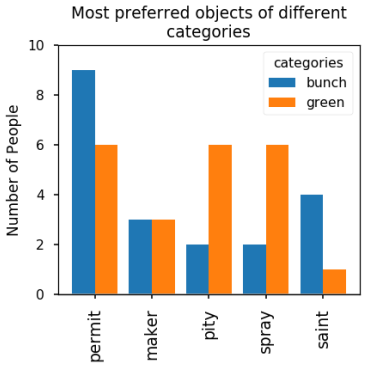

Figure 5. Example Bar Chart from MathVista [19] Associated question: How many people like the least preferred object in the whole chart?

that. We used the Adam optimizer and a cosine learning rate schedule with weight decay of 0.1 (default parameters provided in the finetuning config) through.

3. Training Categories: We conducted experiments where we tuned only on line chart data, only on bar chart data, and on a mix of both to study overfitting effects and the transferability of knowledge from each category.

4. Using Randomization of Options and LLM as a Judge vs not Using it: As noted above, we noticed that the generated data tended to note provide a uniform distribution across all options. We experimented with ablating the randomization post-processing step to understand the impact of not having it. We also ran an ablation removing the final LLM as a judge step to understand how the lower quality raw generated data affected (without filtering) affected performance.

5. Modifying the vision encoder versus keeping it fixed: To understand how much of the improvement we see

Model	GVal-B	TestM-B	TestM-L	TestM-O
Baseline	50.0	64.5	58.3	50.7
LoRA (r = 2)	64.1	-	-	-
LoRA (r = 8)	59.4	-	-	-
LoRA (r = 32)	67.2	61.3	58.3	54.4
Ablate ViT	60.9	61.3	58.3	55.5

Table 2. Only Training on Synthetic Bar Charts

came from improving the vision encoder versus improving the alignment or aligning the language model better, we ablated the finetuning of the vision encoder (as mentioned above, otherwise this component would undergo full fine-tuning along with the language model component).

We kept the batch size fixed at 1 which keeps memory usage minimized but effectively operated at a batch size of 8 by using 8 gradient accumulation steps before updating the parameters. We turned off gradient checkpointing for LoRA (for increased speed of tuning at the cost of additional memory) and kept it for DoRA (otherwise we would run out of memory when tuning DoRA).

6. Results

The main metric we use to measure performance is accuracy on different subsections of the MathVista's testmini, and on a validation set we create using GAMMAS. For testmini, we look at accuracies on the bar chart subsection (TestM-B), line plot subsection (TestM-L), and the other categories in testmini combined (TestM-O). For the validation set we create, we look at either the bar chart subsection (GVal-B), the line chart subsection (GVal-L), or the validation set overall (GVal). For all these datasets, we chose to only evaluate MCQs, as freeform evaluation was challenging, given the many ways to write equally-correct answers.

Tables 2, 3, and 4 show the performance of the baseline and fine-tuned models using different fine-tuning techniques and after being fine-tuned on different subsections of the dataset generated using GAMMAS. Specifically, the tables are split up by the type of fine-tuning data used, with Table 2 listing models that were fine-tuned on only the Bar Charts subsection of GAMMAS, Table 3 listing models that were fine-tuned only on the Line Plots subsection of GAMMAS, and Table 4 listing models that were fine-tuned using the entirety of GAMMAS. In each table, we list the baseline performance, and performance with the fine-tuning techniques we tried for that subsection of GAMMAS (namely LoRA and ablating the vision encoder fine-tuning - denoted as ViT - for Synthetic Bar Charts, LoRA for Synthetic Line Plots, and Lora and DoRA for the entirety of GAMMAS). Note that the numbers in parentheses next to DoRA and LoRA fine-tuned models represent the rank used.

Model	GVal-L	TestM-B	TestM-L	TestM-O
Baseline	49.4	64.5	58.3	50.7
LoRA (32)	57.1	64.9	62.5	53.8

Table 3. Only Training on Synthetic Line Plots

Model	GVal-B/L	TestM-B	TestM-L	TestM-O
Baseline	50.0/49.4	64.5	58.3	50.7
LoRA (32)	68.8/63.6	67.7	70.8	54.2
DoRA (32)	71.9/58.4	71.0	70.8	54.6

Table 4. Training on Both Synthetic Bar and Line Plots

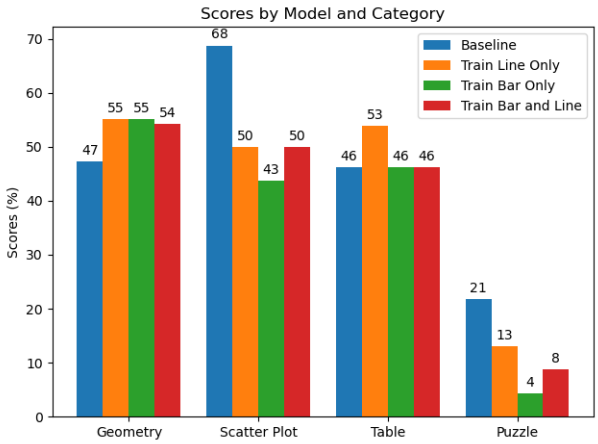

Figure 6. A Fine-grained analysis of TestM-O

We also conduct a more fine-grained analysis of TestM-O, the non-bar-and-line-chart subsection of testmini, in Figure 6.

In Figure 7, we generate a t-distributed stochastic neighbor embedding (t-SNE) visualization, a technique that lets us create a 2D mapping in a way that preserves neighbor identities [12]. To do so, we ran the vision encoder on the just the image from each training data sample, averaged the $1225 = 35x35$ vision embeddings that were output, and passed it through a standard t-SNE algorithm with a perplexity of 30.

In addition to the experiments covered by the tables above, we note the results from the following experiments:

1. Learning Rate Tuning: We found that the learning rate did not have a significant impact on model performance within the range that we tuned.

2. Randomization: We found that ablating the randomization step of our post-processing pipeline did not result in any performance gain or drop; however, we keep this step because it is important to guard against inherent biases that could be introduced by the non-randomness of LM output.

3. LLM as a judge: We found that this was very useful - the validation accuracy for "LoRA (32)" trained on

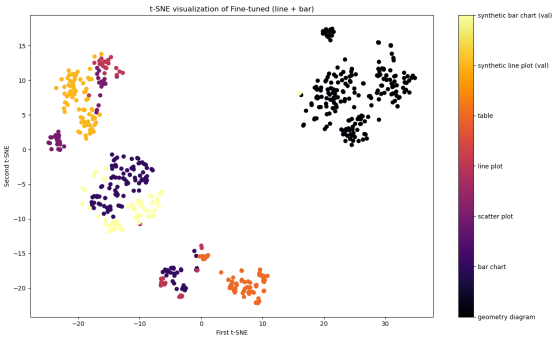

Figure 7. t-SNE visualizing the embedding similarities between different figure types within MathVista, as well as our synthetic generated data (using the val set).

bar chart data degraded by up to 5.1% without the additional LLM as a judge step to filter out bad examples.

We also make the following observations about the results in the tables, which we will discuss in more depth below:

1. Using the highest tried rank of 32 for LoRA parameters resulted in maximal accuracy on our validation set when training on bar chart data.

2. Ablating the fine-tuning of the vision encoder results in a large 9.1% drop in our accuracy on the validation set.

3. If we train only on synthetic bar chart data, we do not notice improvement on the bar chart questions in MathVista. However, if we train only on synthetic line chart data, we notice improvements on the line chart questions in MathVista. If we train on both sets of data together, we notice improvements in both categories of MathVista questions. In all cases where we fine-tune, we slightly improve performance on questions that are not bar or line chart related but in the MathVista dataset.

4. DoRA with r = 32 is comparable with LoRA with r = 32 (from our validation sets, we saw that they were slightly better for bar charts, and worse for line plots). Both outperform the baseline model on the MathVista dataset (both in the bar chart and line chart categories, as well as others).

7. Discussion

GAMMAS Difficulty Level Generally, we observe that models perform worse on GVal than they do TestM-B or TestM-L. For example, in Table 2, we see a stark 14.5% difference in the performance of the baseline model, with TestM-B being significantly easier than GVal-B. We see a similar pattern within Table 3, with the baseline achieving a 8.9% lower accuracy on GVal-L than on TestM-L.

Performance After Fine-tuning, Bar and Line Chart Tasks When fine-tuning with bar charts, we see that performance on GVal-B increases significantly. The LoRA fine-tuned model (with rank 32) achieves the best performance, showing a 17.2% increase. Despite this, performance on TestM-B actually drops by 3.2% after fine-tuning. We hypothesize two reasons for this. The first explanation could be simple overfitting. As we see in Table 4, when we fine-tune with both bar and line charts, performance improves on both TestM-B and GVal-B, not just GVal-B, as previously observed.

However, this pattern is not observed with line charts. In line charts, we see that LoRA fine-tuning (with rank 32) leads to a 7.7% increase in performance on GVAL-L and a 4.2% increase in performance on TestM-L. Line charts had 60 fewer data points than bar charts, which could explain the lack of overfitting.

One reason for this could be that performance on bar charts for these models is already quite good, whereas performance on line plots is generally lower for the baseline model itself. Therefore, it could be easier to improve performance on line charts without overfitting.

Another reason for the observed drop in performance for bar-chart-fine-tuned models on TestM-B could be because of the difficulty of the GAMMAS fine-tuning set. Work in curriculum learning has shown that the order in which models see increasingly difficult examples impacts training convergence and model performance a lot [1, 9]. Because we did not tag examples with difficulty, it is likely that the model saw training examples in an unideal order, limiting its ability to actually learn during finetuning. However, when we finetune with both bar and line charts, we note a performance increase of 6.5% and 12.5% on TestM-B and TestM-L, respectively,

Performance After Fine-tuning, Other Tasks Across tables 2, 3, 4, we see an overall increase in performance on TestM-O, regardless of whether we train using the bar- or line-chart split (or both). This demonstrates that the model's reasoning abilities improved overall, despite only seeing data from a narrow range of skills. As shown in Figure 6, fine-tuned models specifically perform better on geometry tasks, as they require mathematical reasoning that doesn't counter what the models learning during fine-tuning. The performance on scatter plot tasks decreases, which is surprising because it is a related task. Possibly, this could be because while the linguistic questions about the scatter plots were similar to those for line- and bar-charts, the scatter plots themselves were visually different, leading the

model to misreason. Puzzles are both visually and linguistically different from bar and line charts, which is likely what causes the performance decrease observed. Performance on table tasks generally stays unchanged, likely because of the difficulty of table-related tasks for VLMs [19].

Evaluating GAMMAS Faithfulness In Figure 7, we measure how data clusters to check whether GAMMAS faithfully replicates the data distribution in the MathVista dataset. We see that each category of image clusters with other images from its category, which is expected. We also see that examples generated using GAMMAS (indicated by synthetic bar chart and synthetic line plot) generally cluster around the MathVista bar charts and line plots, respectively.

Interpretability: Impact of Vision vs Language We also wanted to understand whether it is improvement in the visual component that results in the overall improvements that we see or improvements in the language component. One datapoint we see above is that, when purely training on bar chart data, not finetuning the vision encoder results in worse performance on our val dataset but comparable performance on MathVista. This might suggest that fine-tuning the vision encoder contributes to some degree of overfitting to the structure of our images, which means that we might reap benefits from using additional vision transforms/adding more noise or variety to our input. We also extracted the embeddings after fine-tuning, and compared them to the ones before fine-tuning by running t-SNE with both sets to understand if there was a noticeable difference caused by fine-tuning in the clustering of these embeddings (i.e. does the same datapoint's embedding change a lot before and after fine-tuning?). However, we did not see this to be the case (most embeddings after fine-tuning stayed relatively close to where they were before fine-tuning on a t-SNE plot) - we omit the figure for this because it is not too instructive (as the points after fine-tuning look very similar to the figure above before fine-tuning).

8. Conclusion/Future Work

In this work, we present GAMMAS: a novel pipeline to generate synthetic multi-modal data for math problem-solving. We synthesize bar- and line-chart-related math problems using this pipeline. Then, we fine-tune InternLM-XComposer2-VL-1.8B using different subsections of our dataset with PEFT techniques like LoRA and DoRA. Generally, the fine-tuned models not only improve on the task we fine-tuned them with, but also on other tasks, with a 17.2% performance improvement being observed on bar chart tasks.

In the future, we are interested in using this pipeline to explore the impact the type of diversity in finetuning data has on model performance. Because the prompting used in GAMMAS has a piece for linguistic and visual diversity, we want to experiment how model performance is impacted when we generate data by omitting either (or both) of the types of diversity we prompted for. We also would like to build a robust evaluation pipeline that would also let us test our models on questions with freeform answers. We noted earlier that during data synthesis, the LLM frequently generated the MCQ answer to be B. We noted a similar issue with Yes/No questions, where the answer was Yes 75% of the time. In the future, we would like to address this by using prompt rewriting to even out the distribution of Yes and No answers.

Lastly, while we only explore parameter-efficient fine-tuning techniques in this paper, we'd like to explore representation fine-tuning (ReFT) techniques in the future as well. While fine-tuning, we only focused on fine-tuning the vision encoder and the LM, but did not fine-tune the sampler, which connects the encoder to the LM. We expect that fine-tuning the sampler would improve model performance even more.

Generating synthetic visual data is still challenging, as leading vision models face noticeable gaps in performance. By harnessing the linguistic/coding capabilities of LLMs to programmatically generate diverse multi-modal data, the GAMMAS pipeline opens up a future avenue for work. While we only generated synthetic bar- and line-charts, data from other tasks like tables, geometry problems, scatter plots, etc. can be synthesized by harnessing code-generation for matplotlib-like tools.

9. Appendices

10. Contributions and Acknowledgements

Shubhra created the data generation pipeline for bar charts, which Ram adapted to line charts. He implemented both the PeFT techniques mention in the paper. He also implemented the LLM-as-a-judge + randomization techniques discussed in the data generation pipeline, and also evaluated the ground-truth results we show to prove the efficacy of LLM-as-a-judge for our use case.

References

[1] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[2] F. Bordes, R. Y. Pang, A. Ajay, A. C. Li, A. Bardes, S. Petryk, O. Mañas, Z. Lin, A. Mahmoud, B. Jayaraman, M. Ibrahim, M. Hall, Y. Xiong, J. Lebensold, C. Ross, S. Jayakumar, C. Guo, D. Bouchacourt, H. Al-Tahan, K. Padthe, V. Sharma, H. Xu, X. E. Tan, M. Richards, S. Lavoie, P. Astolfi, R. A. Hemmat, J. Chen, K. Tirumala,

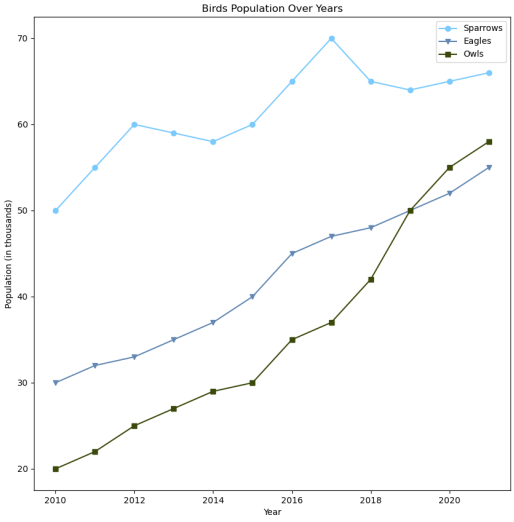

Figure 8. Example Line Chart Generated Using GAMMAS. Associated question: In which year, the population of Eagles surpassed the population of Sparrows if ever?

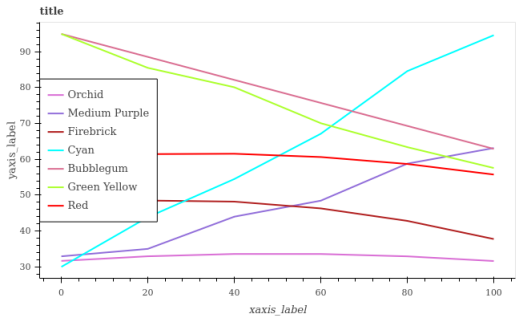

Figure 9. Example Line Chart from MathVista [19]. Associated question: Does medium purple have the second highest value?

R. Assouel, M. Moayeri, A. Talattof, K. Chaudhuri, Z. Liu, X. Chen, Q. Garrido, K. Ullrich, A. Agrawal, K. Saenko, A. Celikyilmaz, and V. Chandra. An introduction to vision-language modeling, 2024.

[3] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

[4] Z. Cai, M. Cao, H. Chen, K. Chen, K. Chen, X. Chen, X. Chen, Z. Chen, Z. Chen, P. Chu, X. Dong, H. Duan, Q. Fan, Z. Fei, Y. Gao, J. Ge, C. Gu, Y. Gu, T. Gui, A. Guo, Q. Guo, C. He, Y. Hu, T. Huang, T. Jiang, P. Jiao, Z. Jin, Z. Lei, J. Li, J. Li, L. Li, S. Li, W. Li, Y. Li, H. Liu, J. Liu, J. Hong, K. Liu, K. Liu, X. Liu, C. Lv, H. Lv, K. Lv, L. Ma, R. Ma, Z. Ma, W. Ning, L. Ouyang, J. Qiu, Y. Qu, F. Shang, Y. Shao, D. Song, Z. Song, Z. Sui, P. Sun, Y. Sun, H. Tang, B. Wang, G. Wang, J. Wang, J. Wang, R. Wang, Y. Wang, Z. Wang, X. Wei, Q. Weng, F. Wu, Y. Xiong, C. Xu, R. Xu, H. Yan, Y. Yan, X. Yang, H. Ye, H. Ying, J. Yu, J. Yu, Y. Zang, C. Zhang, L. Zhang, P. Zhang, P. Zhang, R. Zhang, S. Zhang, S. Zhang, W. Zhang, W. Zhang, X. Zhang, X. Zhang, H. Zhao, Q. Zhao, X. Zhao, F. Zhou, Z. Zhou, J. Zhuo, Y. Zou, X. Qiu, Y. Qiao, and D. Lin. Internlm2 technical report, 2024.

[5] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems, 2021.

[6] X. Dong, P. Zhang, Y. Zang, Y. Cao, B. Wang, L. Ouyang, X. Wei, S. Zhang, H. Duan, M. Cao, W. Zhang, Y. Li, H. Yan, Y. Gao, X. Zhang, W. Li, J. Li, K. Chen, C. He, X. Zhang, Y. Qiao, D. Lin, and J. Wang. Internlm-xcomposer2: Mastering free-form text-image composition and comprehension in vision-language large model. *arXiv preprint arXiv:2401.16420*, 2024.

[7] X. Dong, P. Zhang, Y. Zang, Y. Cao, B. Wang, L. Ouyang, X. Wei, S. Zhang, H. Duan, M. Cao, W. Zhang, Y. Li, H. Yan, Y. Gao, X. Zhang, W. Li, J. Li, K. Chen, C. He, X. Zhang, Y. Qiao, D. Lin, and J. Wang. Internlm-xcomposer2: Mastering free-form text-image composition and comprehension in vision-language large model, 2024.

[8] R. Eldan and Y. Li. Tinystories: How small can language models be and still speak coherent english?, 2023.

[9] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu. Automated curriculum learning for neural networks, 2017.

[10] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. D. Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, H. S. Behl, X. Wang, S. Bubeck, R. Eldan, A. T. Kalai, Y. T. Lee, and Y. Li. Textbooks are all you need, 2023.

[11] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.

[12] G. Hinton and S. Roweis. Stochastic neighbor embedding. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, page 857–864, Cambridge, MA, USA, 2002. MIT Press.

[13] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models, 2021.

[14] A. Q. Jiang, W. Li, and M. Jamnik. Multilingual mathematical autoformalization, 2023.

[15] S. E. Kahou, A. Atkinson, V. Michalski, Á. Kádár, A. Trischler, and Y. Bengio. Figureqa: An annotated figure dataset for visual reasoning. *ArXiv*, abs/1710.07300, 2017.

[16] R. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi. MAWPS: A math word problem repository. In K. Knight, A. Nenkova, and O. Rambow, editors, *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California, June 2016. Association for Computational Linguistics.

[17] B. Liu, S. Bubeck, R. Eldan, J. Kulkarni, Y. Li, A. Nguyen, R. Ward, and Y. Zhang. Tinygsm: achieving ¿80

[18] S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.-T. Cheng, and M.-H. Chen. Dora: Weight-decomposed low-rank adaptation, 2024.

[19] P. Lu, H. Bansal, T. Xia, J. Liu, C. Li, H. Hajishirzi, H. Cheng, K.-W. Chang, M. Galley, and J. Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts, 2024.

[20] A. Masry, D. X. Long, J. Q. Tan, S. Joty, and E. Hoque. Chartqa: A benchmark for question answering about charts with visual and logical reasoning, 2022.

[21] T. N. Mundhenk, G. Konjevod, W. A. Sakla, and K. Boakye. A large contextual dataset for classification, detection and counting of cars with deep learning, 2016.

[22] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph. Gpt-4 technical report, 2024.

[23] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021.

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[25] D. Saxton, E. Grefenstette, F. Hill, and P. Kohli. Analysing mathematical reasoning abilities of neural models, 2019.

[26] P. Song, K. Yang, and A. Anandkumar. Towards large language models as copilots for theorem proving in lean, 2024.

[27] C. Team. Chameleon: Mixed-modal early-fusion foundation models, 2024.

[28] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.